SCHOLAR Study Guide

SQA Higher
Computing Unit 2
Software Development

David Bethune
Heriot-Watt University

Andy Cochrane
Heriot-Watt University

Tom Kelly
Heriot-Watt University

Ian King
Heriot-Watt University

Richard Scott
Heriot-Watt University

Heriot-Watt University

Edinburgh EH14 4AS, United Kingdom.

First published 2004 by Interactive University

This edition published in 2007 by Heriot-Watt University

SCHOLAR Study Guide Unit 2: Computing

1. Computing

Printed and bound in Great Britain by Graphic and Printing Services, Heriot-Watt University, Edinburgh.

Acknowledgements

Thanks are due to the members of Heriot-Watt University's SCHOLAR team who planned and created these materials, and to the many colleagues who reviewed the content.

We would like to acknowledge the assistance of the education authorities, colleges, teachers and students who contributed to the SCHOLAR programme and who evaluated these materials.

Grateful acknowledgement is made for permission to use the following material in the SCHOLAR programme:

The Scottish Qualifications Authority for permission to use Past Papers assessments.

The Scottish Executive for financial support.

Contents

Topic 1

Introduction to Software Development Process

Contents

Prerequisite knowledge

There are no prerequisites for this introductory topic.

Learning Objectives

- *understand what is meant by 'software engineering'*
- *understand what is meant by 'software development'*
- *recognise the professional issues that are involved in the development of a computer system*
- *understand the need for software development*
- *understand what is meant by well planned programs.*

1.1 Introduction

In this topic you will be introduced to the Software Development Process and the reasons why such approaches to the topic exist. The subject has its roots in software engineering and this is discussed from a historical perspective where the magnitude of the processes involved are made clear. This leads on to the need for well-planned programs together with the associated problems and preferred solutions.

1.2 Overview

Computer systems are now an important aspect of everyday life. Indeed it would be difficult to visualise a world without the day to day impact that computers have on our lives: cash machines, supermarket tills, petrol pumps, travel tickets, payslips and bills, the Internet and e-commerce and so on.

In the 1960's - 70's when mainframe computers were at the heart of large commercial and industrial computer systems, mostly for batch processing tasks, companies relied on vast teams of programmers to implement software solutions that required system specifications. Today's computing systems offer much more complex scenarios and their development requires well thought out plans and sophisticated software tools for defining each stage of the process in a much more formal manner than previously attained.

The first few topics of this unit describe the development of software. The development of software follows a definite process, known as the **software development process** (SDP). It introduces a few of the models which are used to represent the software development process and describes in detail the **traditional model**.

A model allows users to understand the development process and the stages involved in the development sequence. The model aids management and good management helps produce a higher quality product, possibly in a shorter period of time and at less cost. In the last thirty years or so a number of models have evolved, each with particular strengths and weaknesses, to make the analysis and design stages of software development more manageable.

Before looking at some of these models we should take a brief look at why software development evolved as an engineering process. This is best understood by looking at the sorts of things that have gone wrong in large software developments.

1.2.1 Software engineering

In this topic we want to consider good practice in developing programs within the **software development environment**. To do so we think it is useful to put software development into a wider context, as it is part of a subject known as **software engineering**.

The ideas behind software engineering are to provide an approach to the development of software by designing and building software systems that are:

- of high quality
- cost effective
- produced to specification
- delivered on time

Rather than produce software on an arbitrary basis, the idea is to bring together a variety of tools, techniques and methods which will help create *reliable* software. By these means programs can be designed to be *maintainable* - that is they are presented and documented so clearly that they can be updated by another programmer at some future time.

1.2.2 Software development

There are many reasons why software development is required. Some include:

1. increased complexity and sophistication of computing systems
2. escalating costs of software systems
3. unreliable software systems produced without planning
4. poorly performing software
5. difficulty in maintaining the software

If you have been involved in any computer systems before you started this course, you are probably nodding your head, thinking of all the computing disasters you have seen. But if you are new to computing, then maybe the table in the next topic might help you to see the differences between real-life programs and the sort of programs you are (or will be) writing. We think that if you understand the problems of large-scale programming, the need for software development will be obvious. Without this understanding, it is all too easy to dismiss software development as a not-very-useful exercise, but one which will give you a Higher credit.

It may seem to you that software development on this scale is overkill for the sort of programs you are going to be writing. In many ways it is, but then it is easier to learn about software development by tackling a small, straightforward task than by starting out on a mammoth project. Mistakes are easier to spot and easier to fix at this stage: software problems in a commercial or industrial system are likely to be 100 to 500 times more costly to find and repair once the software has been installed and running.

1.3 Real-life programs and classroom programming

What are the differences between the two? Table 1.1 should give you an indication

Table 1.1: Real programs vs classroom programs

Real-life Programs	**Classroom programs**
...are large and complex. It is difficult to hold their details in your mind. Because of this you have to specify them formally.	...are small and simple. They can be described in a few sentences and probably understood in minutes.
The main difficulty is with (a) understanding the problem; (b) ensuring that your understanding actually matches the users' requirements; (c) designing a solution.	The main difficulty you have is with writing the solution in a programming language which you are just learning, and getting your program to work correctly.
Development time is long. Maybe as long as 5 years. It is impossible to remember all aspects of a program for this length of time, so you have to document the design decisions and all your changes.	Development time is short. You should be able to get the core of the solution working in a few hours. Consequently you can hold details of all aspects of the program in your head. Documentation is skimpy or non-existent.
Testing is going to be long, intensive and exhaustive. It will be done by a separate team whose function is solely to test programs and find faults.	Your testing is likely to be fairly basic.
A program's lifetime may be decades, and it may undergo changes (called 'maintenance'). The maintenance is probably going to be done by someone who has not written the original program and knows nothing about the original design decisions unless they have been documented.	The life of the program is short - only long enough for you to pass! Similarly, the documentation only has to stand up to the examiner's scrutiny.
Programs are written for use by other people who might have little understanding of computers.	The programs are not intended to be used.

Real-life programs are written for use by **clients** who require the software systems to help them do their jobs. They probably expect them to work like any complicated electrical appliance such as a TV. You switch it on and it works. Change channels and the channels change. The brightness and other controls do what they are supposed to do, without stopping the TV from working or having some unexpected side-effects.

The programmers' goal is to produce software as robust and easy to use as a TV. In reality this is going to be impractical, but when you design programs this should be something you need to bear in mind. For example, keyboard input should be checked and incorrect input rejected. But this is not enough; you need to give a message to the user to help them type the correct information next time. If you do not do this, you leave them floundering; they know what they typed is wrong, but have no idea why or what the correct input should be. Any other errors which arise in the program should

be clearly notified to the user in common-sense language. A Windows 'exception error' an example of an intimidating and aggravating error message which provides no useful information to anyone except a dyed-in-the-wool Windows programmer.

1.4 Computing disasters

It has been said *"Computers make very fast, very accurate mistakes"*. While this may be undeniably true it has to be said that complacency still exists in the use of computers and the programs running on them.

Software development is important because in its absence you are bound to have a disaster in a project of any great size. Large projects demand a large investment of money, time and resources before any returns are possible.

Consider the following:

- One in five software systems fail to deliver to the agreed specification
- Software maintenance is the single highest computer-related cost for many companies
- Some systems can take many years to develop running the risk of obsolescence before they are even commissioned.

Naturally a fair bit of planning goes into them. But perhaps the planning is not all it might be because, even with the best will in the world, things can go terribly wrong. The list below contains a few examples of computing disasters that could be attributed to complexity and the need for better planning:

1. **Ariane 501** - on the 4th June 1996 Ariane 501 exploded 40 seconds after takeoff due to a software bug. This was an incidence of data conversion of a too large number!
2. **Y2K Problem - The Millennium Bug** - problems with the way the date was stored on computers
3. **NASA Mars Lander** - problems in software occurred when different measurement units were used - there was confusion between pounds and kilograms
4. **Home Office Immigration System** - Ordered in 1996 and supplied eighteen months late by Siemens in 1999 at a cost of £80 million. It was supposed to speed the processing of asylum claims up but could not cope with the backlog and did the very opposite. It was finally scrapped in 2001. Government ministers were blamed for ordering the "over-complex" system in the first place
5. **London Ambulance Service** - in 1992 they took on a despatch system that failed calamitously. Ambulances were sent to the wrong place, did not arrive when expected and the system generally caused major disruption to patient care and services
6. **Passport Agency** - in 1999 a new system (Siemens again) that worked much less efficiently than the system it replaced. Delays went up from two weeks to seven,

with a backlog of more than half a million passports. This cost the taxpayers £12 million and forced many people to cancel holidays. The Government increased passport charges to recoup wasted money from this system

7. **Post Office swipecard system** - a one billion pound project that ICL were to install in throughout Britain. It began in 1996 and was stopped in 1999, having been, as the official report said, "blighted from the outset"
8. **French national library** - received a new system (two years late and 40% over budget) which worked so badly that librarians walked out
9. **ATMs in Germany** - during the changeover from Marks to Euros in 2002 a programming error in the banking system allowed people to withdraw any amount cash by typing in an arbitrary PIN code.
10. **London Millennium Bridge** - in 2000 the newly built Millennium Bridge wobbled when pedestrians attempted to walk over it. In the computer simulation the programmers had used the wrong estimates for pedestrian forces.

Of course, all these systems were planned, but with better planning, these mishaps might have been prevented. Without planning, probably all large projects would go wrong.

The most basic notion of well planned programs is producing software which does what the user wants. This is easy to say, but more than a little difficult to do. If you are taking this course at the same time as learning a programming language, then we are sure you know this is so from your own experience.

1.5 Information sources on computing disasters

The World Wide Web and Internet links go out of date rather quickly. You might want to try accessing the following sites which are current at the beginning of 2004.

1. **Ariane 501**
   - A Bug and a Crash by James Gleick, http://www.around.com/ariane.html
   - From champagne to shock: Ariane 5 explodes - Science Online, http://www.flatoday.com/space/explore/stories/1996/060496c.htm
   - Ariane-5: Learning from Flight 501 and Preparing for 502 - European Space Agency (ESA), http://esapub.esrin.esa.it/bulletin/bullet89/dalma89.htm
2. **Y2K Problem - The Millennium Bug**
   - Looking at the Y2K Bug - CNN, http://www.cnn.com/TECH/specials/y2k/
3. **NASA Mars Lander**
   - NASA: Human error caused loss of Mars orbiter - CNN (including video of news reports), http://www.cnn.com/TECH/space/9911/10/orbiter.03/
   - Metric mishap caused loss of NASA orbiter - CNN, http://www.cnn.com/TECH/space/9909/30/mars.metric.02/

4. **Passport fiasco**
    - http://news.bbc.co.uk/1/hi/uk_politics/486821.stm
5. **Asylum issue**
    - http://news.bbc.co.uk/1/hi/uk_politics/1171147.stm
6. **Millennium bridge**
    - http://www.arup.com/MilleniumBridge

Some other general web sites on disasters are:

Risks Digest: http://catless.ncl.ac.uk/risks

Horror stories: http://www.cs.tau.ac.il/~nachumd/verify/horror.html

No bugs in Win 95 - Interview with Bill Gates: http://www.cantrip.org/nobugs.html

One of the best books on computing disasters is *The Mythical Man-Month* by F.P. Brookes, Addison Wesley, 1995. This describes a huge IBM disaster in the late 1960s - early 1970s, but is just a relevant for today. It is also a very good read - not filled with jargon and tiresome self-justification. Well worth getting.

Investigating computing disasters

Several WWW sources of information about computing disasters have been given above. For others we have not given any site details. Try to find out more about any two (or more) that might interest you. Alternatively try some of the general links to find two (or more) which are not in the list. Write a summary of two disasters that you have found using around 500 words in total for both.

1.6 Well planned programs

The most basic notion of well planned programs is to produce software that does what the user wants. This is easy to say, but more than a little difficult to do. If you have been involved in using a programming language to solve a problem, then we are sure you know that this is the case from your own experience.

Other characteristics of well planned programs are:

1. the software should be maintainable: software with a long lifetime is likely to need changing. This means that software must be written and documented in a way which makes change simple and straightforward
2. the software should be reliable
3. the software should be efficient. It should not make unreasonable demands on the hardware on which it will run. But higher efficiency can lead to less maintainable software as programmers use shortcuts which are effective, but difficult to understand
4. the software should have an appropriate user interface. 'Appropriate' means that

you need to consider the background and capabilities of intended system users.

There is often a trade-off between these factors and the cost of producing the software. This is illustrated in the diagrams in Figure 1.1.

As you can see, costs increase dramatically as developers attempt to provide software that is 100 percent reliable. Making software more efficient also increases the amount of money that needs to be invested.

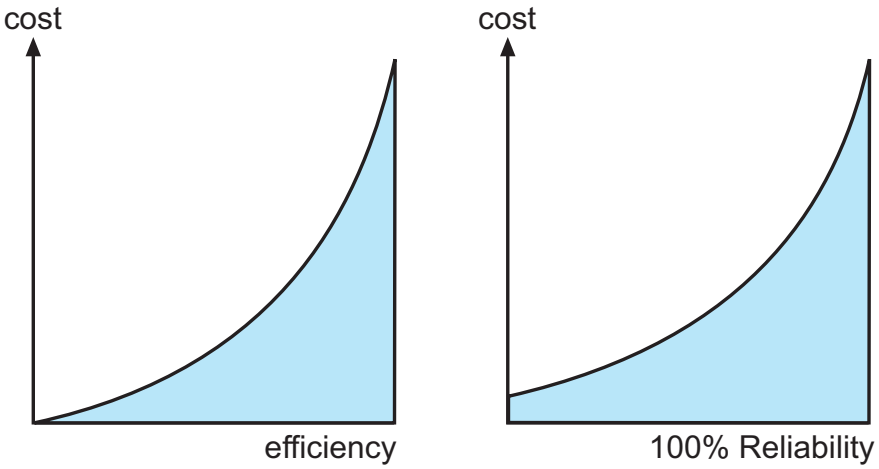


Figure 1.1: Cost of Software Systems

1.7 Summary

The following summary points are related to the learning objectives in the topic introduction:

- software development has its roots in software engineering;
- software development is a complex process involving many facets of professional expertise;
- no matter how simple or complex a program is the elements of software development are almost mandatory to ensure well-planned programs;
- well-planned programs take time to evolve;
- no matter how rigorous the software development processes are, errors can still occur in large, commercial software systems.

1.8 End Of Topic Test

An online assessment is provided to help you review this topic.

Topic 2

Features of Software Development Process

Contents

Prerequisite knowledge

Before studying this topic you should be able to:

- *describe the following stages of the software development process, analysis, design, implementation, testing, documentation, evaluation, maintenance;*
- *describe and be able to use test data (normal, extreme and exceptional);*
- *describe the features of a user guide and a technical guide;*
- *evaluate software in terms of fitness for purpose, user interface and readability.*

Learning Objectives

- *understand the iterative nature of the software development process*
- *understand each stage of software development*
- *identify various models of the software development process*
- *understand the purpose of the software specification and its status as a legal contract*
- *understand and be able to describe corrective, adaptive and preventative maintenance*
- *understand the need for documentation at each stage of the software development process*

Revision

Q1: The software development process consists of seven stages. Three of the stages are analysis, testing and maintenance. Which one of the following statements correctly identifies the missing stages, in order?

a) Design, documentation, evaluation, implementation
b) Design, evaluation, documentation, implementation
c) Design, implementation, documentation, evaluation
d) Design, documentation, implementation, evaluation

Q2: The software development process is an iterative process. This means that (choose one):

a) Certain stages of the process are re-written to improve quality
b) Certain stages of the process are re-visited to make sure all is well
c) Certain stages of the process are ignored to save time
d) Certain stages of the process are very complex so more time is spent on them

Q3: Which one of the following statements refers to an essential item of documentation that should be produced during the software development process?:

a) User Guide
b) Technical guide
c) Structured listing
d) All of these

Q4: One of the characteristics of a computer program is the user interface. Which one of the following is NOT a part of this aspect?:

a) Plenty of colour
b) Help screens
c) Instruction screens
d) Screen layout

Q5: Which one of the following statements concerning a structured listing is true?:

a) It makes the program run faster
b) It is part of the design stage of the software development process
c) It is used to spot possible errors in a program
d) None of the above

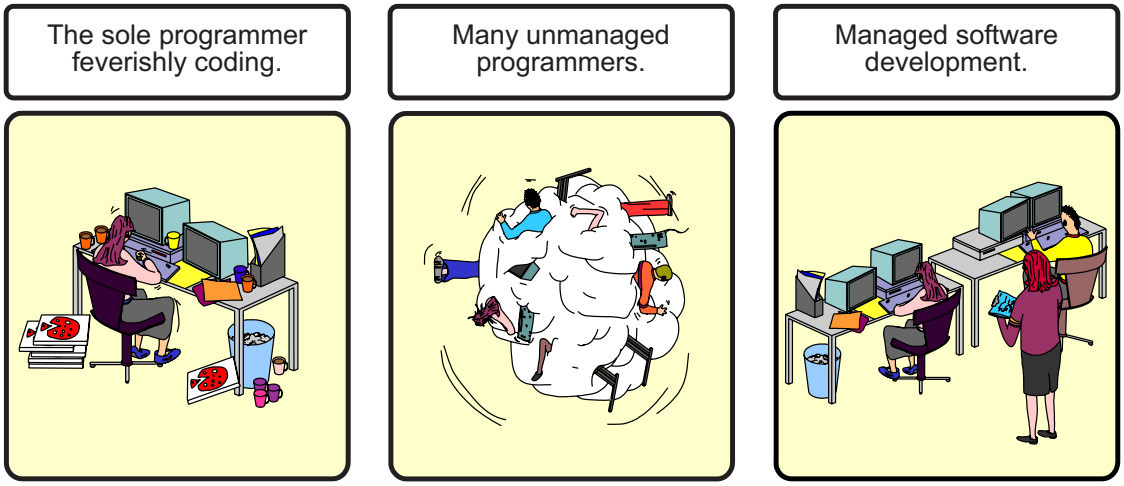
2.1 Introduction

This topic introduces the various mechanism of the software development process. From the initial client specification to the production of a working program can take considerable time and effort by the development team. The process involves constant revision and evaluation at every phase of the project which makes it predominantly an iterative process. This ensures quality and efficiency in the final product. Various models are introduced that aid the software development process but you will find that the perfect solution does not exist.

2.2 Preamble

An individual may write a program for personal use. If it does not work then it can be changed. If more features or facilities are required then the individual can make amendments to their program.

This ad hoc method is not satisfactory in commercial environments where the goal is the creation of large scale software. A more structured approach is necessary. Some of the reasons are concerned with the commercial nature of the business, others you have already met in previous topics.



The organisations creating software usually do so for profit. Money, time, and people are involved. The people involved have different points of view; some are clients, wanting to buy software; some are developers concerned in creating software. Managers are concerned with efficiency and profit within their organisation.

In the development of software, the three aspects which the developer must consider are:

- data
- processes
- human computer interface

In traditional structured design, the primary tasks are to focus on the processes. A **process** is the work that a program carries out on data or in response to certain inputs.

The software development process does not always start in the same place. Sometimes there will only be an outline of the problem. At other times, a **specification** will be available.

The specification must be agreed with the clients. Work on the problem and the solution is often carried out by a group of people, called the **development team**.

The aim of the team is to produce a new software system that will solve the problem.

Software houses are in business to produce software and sell this to the world at large. They develop software for sale to a market which they believe exists. They also write

bespoke software to meet specific needs.

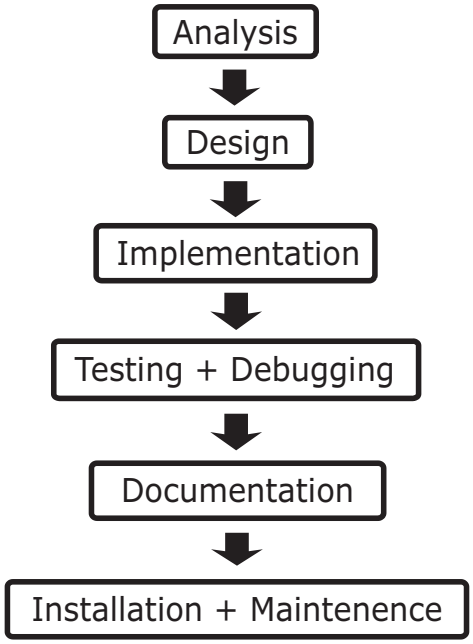
These materials are concerned with projects that have definite clients; however reference will be made to software development as it occurs in software houses.

2.3 The Waterfall Method

The traditional software development process contains a sequence of stages. The precise names of the stages, and even the number of stages, are not universally agreed. They differ from book to book and from developer to developer. Much depends on the aspects of software development that a book wishes to emphasise or that a particular developer prefers.

Referred to as the **waterfall model** this was one of the first models designed for software engineering, which arose as a response to the disorganised ways in which previous software systems evolved.

In this course we will consider the following stages:



The idea here is that the development of the system flows down or cascades through the stages like water flowing down a fall. As each stage is completed responsibility and control is passed down until the final section is completed. We will look at each stage in turn very shortly.

An important aspect of the software development process is that it is an **iterative** process. An iterative process is one that incorporates **feedback** and involves an element of **repetition**. The reasons for this will soon become apparent. You will meet iteration again in later topics dealing with the software development environment.

As you can see one of the main drawback to this model is the fact that if any unnoticed errors occur at any stage of the process then the entire model has to be revised to take

these errors into account

The model, as it stands, represents an ideal world rather than reality. Developers do not know everything the client will need at the start of a project; they make wrong decisions, possibly based on incomplete information. Even with perfect knowledge and infallible decisions, the model could not stand as it is because, as time goes by, more information comes to light which changes how the software is to perform.

The model can be improved significantly if it is made to be iterative.

Ideally, you would start a process with the analysis and work through the stages in turn, doing everything only once. In practice, this happens rarely. People make mistakes, faults become apparent that can only be corrected by going back to an earlier stage of the process. The model can be modified to represent this iteration.

Other models exist that are based on the waterfall method such as Structured Systems Analysis and Design Method (**SSADM**). This contains a comprehensive and rigorous set of standards that are implemented at the analysis and design stages only. We will go into this method more fully later in the topic.

2.3.1 Review questions

Q6: In the process of software development the waterfall model emerged because?

a) Software engineering dictated that such a method should exist
b) The model is easy to understand
c) The evolution of software systems were disorganised
d) Using It is an error-free process

Q7: Which one of the following statements regarding the order of the stages in the waterfall model is correct?

a) Design, implementation, testing, documentation
b) Analysis, testing, implementation, design
c) Design, testing, evaluation, implementation
d) Analysis, design, testing, evaluation

Q8: The waterfall model can involve iteration. Which one of the following statements is true?

a) The model will be more realistic
b) Use of the model will become error-free
c) Use of the model will be simpler to understand
d) The model will be easier to implement

Q9: Software developers cannot get the software correct at the first attempt. This is because:

a) Software systems can be very complex
b) Unforeseen errors always creep in that take time to solve
c) There could be problems with the client changing his/her mind
d) All of the above

Q10: Regular feedback of information to members of the development team is important. This is done in order to:

a) To speed up the development process
b) To enable personnel to discuss progress
c) To keep the team happy
d) To lower the costs on a regular basis

Sentence completion - waterfall model

On the Web is a sentence completion task on the waterfall model. You should now complete this task.

2.3.2 The Analysis Stage

This stage (and also design) is extremely crucial to the entire cycle of events. Any problems occurring at this stage will be propagated through the system and will become increasingly costly to rectify when discovered.

Analysis is an attempt to understand a given problem, clearly and exactly, and to generate a solution. The outcome will be a specification that is used as the basis for all subsequent work.

Sometimes, this stage begins with a vague idea or rough outline of the problem and ends with a precise **problem specification**. On other, rather more formal, occasions, it will start with a full **requirements specification** that serves as a **legal contract** between the client organisation and the development team and end with a **system specification**. This will include hardware and software specifications, and notes on project issues such as objectives, constraints, costs and schedule. It may also include a full **functional specification**, which will describe exactly how the system is meant to behave. The functional specification is what the development team will follow in creating the software system.

Questions to be asked at this stage would include:

- What are the new system requirements?
- What are the costs involved?
- How long will it take to implement?

Details would be gathered by a variety of methods such as interviews and questionnaires.

2.3.3 The Design Stage

Once the precise problem specification has been agreed by both client and development team then the design of the solution begins. The design process is methodical, using techniques such as **structure charts** and **pseudo-code**. The problem is approached by breaking it down into a collection of relatively small and simple tasks. This approach is known as **top-down design** or **stepwise refinement**.

These techniques will be discussed more fully later on.

There are certain characteristics which all good software should possess:

- robustness
- reliability
- portability

The development team will attempt to make the design of the program both robust and reliable. In as much as the terms are distinguishable:

- a reliable program is one that does not stop because of faults in its design
- a robust program is one that can cope with errors when it is running.

To put it another way, an unreliable program is one that hangs or crashes for no apparent reason whereas a non-sturdy program is one that cannot cope with events that the world throws at it.

One of the matters for decision at this point is that of the language of implementation. and the Software Development Environment, *eg should the program be written in C++ under Windows or in Java under Unix?* One of the factors which may affect this decision is the portability of the resulting code i.e. can the software be moved to a different hardware platform and still work effectively.

Identifying the characteristics of good software design

On the Web is a interactivity. You should now complete this task.

2.3.4 Review questions

Q11: Which one of the following processes describes breaking a complex system down into more manageable components?

a) Top-down design
b) using pseudo-code
c) refined design
d) prototyping

Q12: In the software development process a legal contract is represented by which one of the following?

a) functional specification
b) problem specification
c) requirements specification
d) systems specification

Q13: The outcome of the analysis stage of software development is to ascertain which one of the following?

a) the main costs of the project
b) time taken to complete the project
c) hardware required to run the system
d) All of the above

Q14: The completed software system should be able to cope with many errors while running. This means that the software is (choose one):

a) Portable
b) Robust
c) Reliable
d) Stable

Q15: Which one of the terms below is included in the **functional specification**?

a) A description of how the program should operate
b) the hardware used to run the software
c) the nature of the problem to be solved
d) an outline of the problem solution

2.3.5 The Implementation stage

At this stage, the **programming team** will make use of **test data**.

This data is designed to check that the program works properly, and that it is reliable and robust. **Testing** is often confused with the **debugging** of a program, but these are not the same, though they are very closely related.

- Testing establishes the presence of faults in a computer program
- debugging is the finding and removing of these faults.

Also at this stage will be included **internal documentation**. This is commentary within the program to explain the various stages and to record any changes that might be implemented in the coding during debugging. One aspect of this is the use of meaningful variable names which we will cover later in the software development environment.

Standard Algorithms

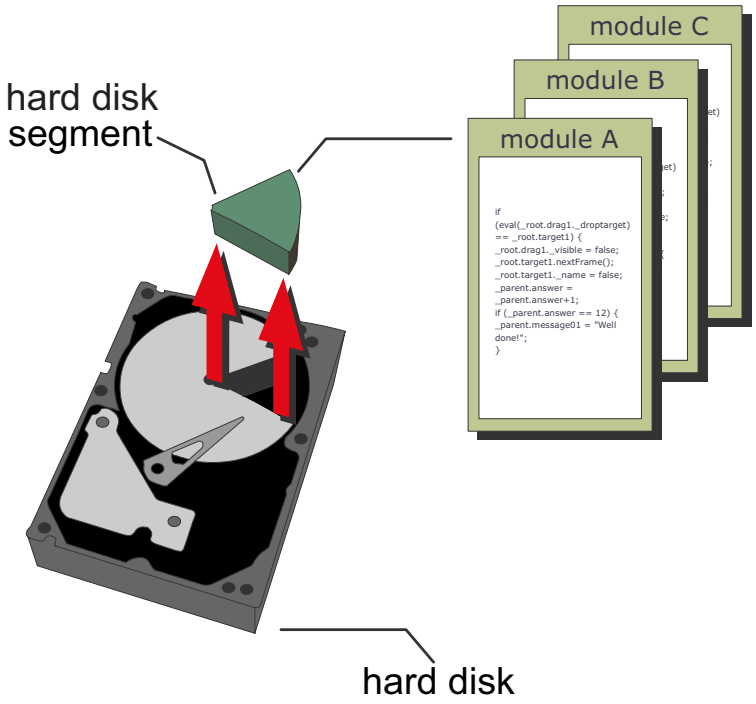
Most projects will use certain standard algorithms. Programmers need to be familiar with these common algorithms. Ones that you will become familiar with later include:

- linear searching
- counting occurrences
- finding maxima and minima

Module Libraries

It is often possible to use, with or without alteration, modules that have been previously written and have been retained in a module library . A module library will include code for

these standard algorithms. Most development environments come with a large library of modules. Programmers can use these in the code they are developing. These libraries will include mathematical functions, modules for converting text to numbers, etc.



2.3.6 The Testing Stage

Testing has several purposes. It should check that:

- the software meets the specification i.e. is correct
- it is robust
- it is reliable.

Testing follows a **test plan** or strategy, involving carefully selected test data, with a view to ensuring that a reliable product has been constructed. Important aspects would be:

- what part of the program is being tested?
- what is the expected output using suitable test data?

Testing can never show that a program is correct. Even with extensive or **exhaustive testing**, it is almost certain that undetected errors exist. Testing can only demonstrate the presence of errors, it cannot demonstrate their absence. A program can be regarded as succeeding if it passes a test; the test can be regarded as succeeding if it makes the program fail.

Commercial software is not exhaustively tested at the testing stage. Software can be complicated and the available time is limited. There must be a balance between creating a product for the market and exhaustive testing. If errors become apparent after release, the company will fix them and release with an updated version.

2.3.7 The Documentation Stage

Users will need to be able to read and learn about the new system. The documentation should include a **user guide** for people who will be using the system, and a **technical guide** for those who will be maintaining it. Commercial software is meant to install itself, more or less, but advice on installation will also be included in either or both of these guides

Other documentation is largely for the benefit of the development team and will include all the documents produced in the course of the development process. This documentation is essential for certain kinds of maintenance or for future revisions of the software. Final documentation will include a **structured listing** of the program

Sentence completion - Documentation

On the Web is a interactivity. You should now complete this task.

2.3.8 The Evaluation Stage

Evaluation is the formal monitoring of a system to ensure that it is performing its purpose accurately, efficiently, cost effectively and in a timely manner. The performance of the system must be matched against a given set of *criteria* such as the initial project specification.

Evaluations of various kinds are an important aspect of the software development industry. Evaluations are used to determine if systems are usable, cost effective, conforming to performance criteria, etc. The basis of evaluation is in social science methods using techniques such as observation, interviews, and questionnaires. Additionally techniques such as automatic data logging are used. Many organisations bring in consultants who design and carry out evaluations as the skills required to carry out effective evaluations are highly specialised.

There is no limit to the number or type of criteria that are used in an evaluation. A very important aspect of planning evaluations is defining the criteria. Listed below are a number of evaluation criteria used in industry:

- the time it takes to install a piece of equipment
- the number of errors an operator makes while doing a specific task
- the time it takes an operator to complete a task
- the number of times a computer crashes or hangs
- the number of phone calls made to a help-line
- the number of times a user consults a manual.

The key criterion in evaluating a software product has to be whether it is **fit for purpose** i.e. does it meet the original specification and allow the client to carry out their tasks? Main questions that may be asked are:

- how closely does the solution match the specification?

- is the solution what the clients were looking for?

Other matters for review will include cost and schedule

Sentence completion - Evaluation

On the Web is a interactivity. You should now complete this task.

2.3.9 The Maintenance stage

This, as a rule, is the most time consuming stage. Software does not wear out but it usually needs subsequent modification. Some bugs or design shortcomings only become apparent over time. In addition changes might have to be made to adapt the system to new demands or legislation (data protection guidelines for example).

It can be tempting to add patches as they are required. However, changes should be made in an organised manner, having regard to the system as a whole and following good practice in software development.

Proper maintenance depends on accurate error reporting from users.

2.3.10 Review questions

Q16: Software testing and debugging are different because (choose one)?

a) Debugging finds faults and testing removes them
b) Debugging takes longer to implement than testing
c) Testing finds faults and debugging removes them
d) Testing takes longer to implement than debugging

Q17: Which statement is true about test data?

a) It makes the program run faster
b) It makes the program more efficient
c) It makes the program run more reliably
d) It makes the programs run error-free

Q18: One of the main aspects of software evaluation is **fit for purpose**. Which one of the statements reflect this?

a) The final program runs error-free
b) The final program meets the original specification
c) The final program is cheaper than originally planned
d) The final program runs more efficiently

Q19: Documentation is an important aspect of the implementation stage. This is because (choose one)?

a) It makes programs more readable
b) It documents what the code does
c) It records changes made to the program
d) All of these

Q20: In the documentation stage of the development process the order of appearance of the documents will be?

a) Structured program listing, installation guide, user guide, technical guide
b) Installation guide, structured program listing, technical guide, user guide
c) User guide, structured program listing, installation guide
d) Technical guide, structured program listing, technical guide, user guide

2.4 Analysis in closer detail

Analysis, sometimes called systems analysis, is the job of a specialist person - the **systems analyst**.

The work that takes place in this stage of the development process varies from case to case. Where only a rough outline of a problem exists, the analyst will have to perform a full systems analysis.

A requirements specification is the starting point and the analyst must proceed from there. The aim is to produce a clear specification that the rest of the development team will use in the subsequent stages.

Full systems analysis has three phases:

1. collection of information
2. analysis of information collected
3. production of a problem specification or user requirements specification.

These ideas are explored further later on under the role of the systems analyst.

2.5 Design in closer detail

The product of the analysis stage is the completed specification. During the design stage, the specification of the problem is used as the basis for a planned solution. The work on the design is carried out by a designer: this might be same person as did the analysis or it may be a member of the programming team. Most analysts can program and many programmers can carry out analysis. People who can do both are described as analyst/programmers.

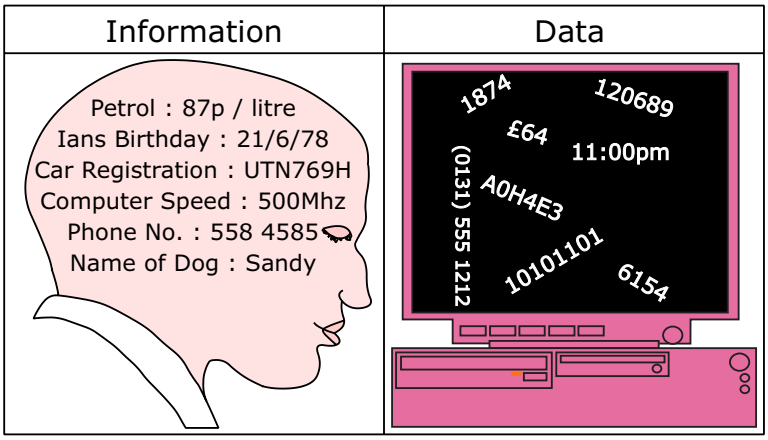
The three major elements of the design are the:

- interface
- data structure
- main program.

2.5.1 The human computer interface

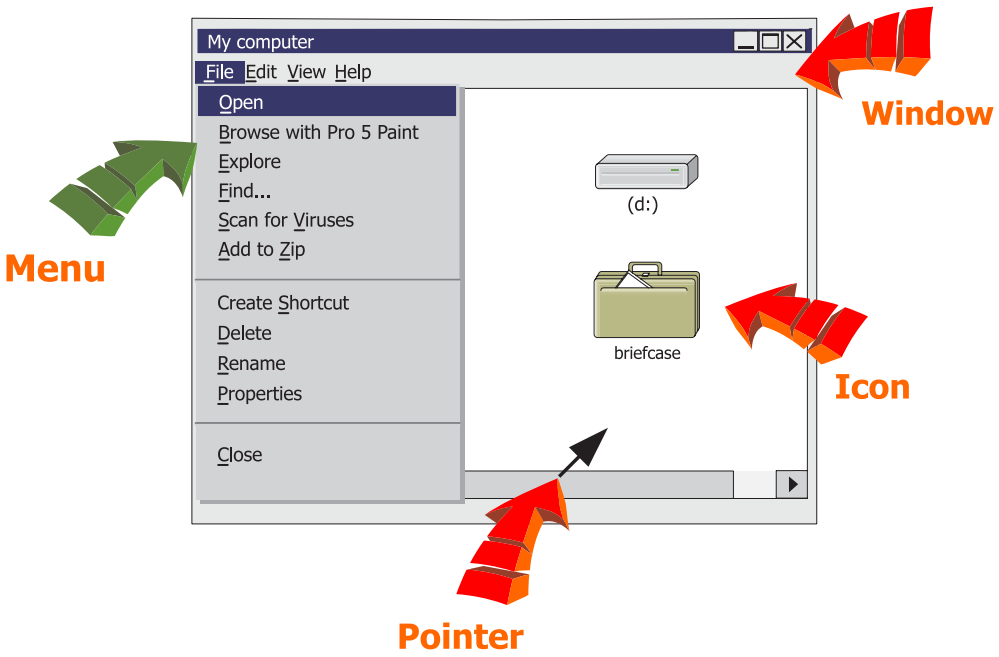
The **human computer interface** is all that a user sees of a program. The HCI can be called the Human Computer Interaction, Man-Machine Interface, user interface or user environment. Design of the HCL is an important aspect of this stage.

A program's viability often relies on the quality of the HCI. A good interface makes things easier for the user. A bad interface can introduce mistakes and cause irritation and impatience.



Modern HCIs are often designed as Graphical User Interfaces (GUIs) which provide a WIMP environment. WIMP stands for Windows Icons Menus Pointers (although some textbooks refer to Windows, Icons, Mouse, Pulldown menus). A GUI provides a set of Windows, which contain Icons and Menus. The user controls the program by means of a Pointer which reflects movement of a mouse, trackerball or other input device.

The WIMP environment



The HCI must allow easy navigation. Users should be able to move from one screen to another in a straightforward manner and to leave screens when they wish. Some sites on the Internet, for example, include a site-map to show the user how different forms are linked.

HCI must be consistent, so that similar actions in different parts of the interface have similar responses throughout the program. Prompts given to the user should be consistent. Different screens should look as though they belong to the same software package.

The HCI should provide on-line help, offering intelligible prompts and send messages and warnings to the user about the consequences of choices made, *eg send a warning if a user chooses to delete data*.

HCI design is based on an appreciation of what the user wishes to see. The designer thinks in terms of the windows (often called forms) that are presented to the user.

Identifying characteristics of a good user interface

On the Web is a interactivity. You should now complete this task.

2.5.2 Data structure

A program must perform operations on the **data** supplied to it. The data should be **structured data**. The choice of data structure will affect the entire program.

A sorted array, for example, can be searched quickly but it is more difficult to maintain or to add data items. The array must be sorted in the first place. Any additions or deletions must leave the array in order and tightly packed. An unsorted array allows additional items to be easily added but searching can be slow.

When the amount of data is likely to be very large, the designer must consider the physical capacity of the hardware. For example, a million records, of a thousand bytes each, will require about a gigabyte. If the clients want these records ordered in different ways in different parts of the program, even a modern PC may have insufficient memory.

Many large systems involve a **database**. In many cases, the data are held in different tables which are linked together. These links are called relations and such a database is known as a relational database . The fields that are to be in the tables and the relations between tables need to be defined at the design stage.

Object oriented design attempts to treat data and **objects** together. An object brings together items of data and the operations that can be carried out on it. For example a data item might be a customer's record, and the operations might include creating, displaying, editing, and deleting.

2.5.3 The Main program

This is the end product of the software development process and represents the efforts of the development team being realised. It may take many months or even years to arrive at a working solution to the initial problem so it must be designed with due care and attention.

The system specification and functional specification will form the basis of designing the program under the following headings:

- hardware specification
- choice of high level language
- how the software will finally behave
- choice of operating system
- portability of the system

Depending on the client's requirements each of the above will have a significant part to play in the final project:

Hardware aspects will include processor speed (does the program require multiple processor facilities for optimum performance as in networked database applications or maybe maximum memory for caching information on a regular basis). How much external storage is required for, say, regular backing up procedures and on what medium? This is an important issue especially on networked systems where users' files are archived on a daily basis.

The choice of high level language will be chosen that is best suited to the problem but also one which the programming team is conversant with and proficient in its use. Modularity will be an important issue where the team can share the workload by compiling modules independently thereby reducing the overall development time. Module libraries can be a rich source of standard algorithms that be used in software projects of many types.

How the system behaves will be determined by the reliability and robustness of the program. Actual and expected outputs should be in agreement as far as possible and this can only be ascertained by rigorous testing at the implementation stage.

Choice of operating system will relate to the functionality and feel of the HCI. Windows might offer much in the way of colourful screens, interactive help and dialogue boxes etc. It must also affect how the program runs and behaves: is the OS software stable enough for the developed program to behave normally without crashing or does the OS offer a true multi-tasking environment, as in UNIX with the added benefits of in-built security. Nowadays Linux is being seen as a viable operating system which is both stable and not difficult to use (cf UNIX).

If the developed software can be moved to different machine architecture and still run to specification then it will be deemed to be portable. In some cases this might not be required but it is a characteristic that good software should possess.

2.6 Implementation in closer detail

If the design has been thorough the implementation should be straightforward. It should be a matter of translating the pseudo-code into code, line by line.

The design is implemented when it has been converted into code which can be used by

the computer system.

The code is written in a **high-level language**, such as Pascal, C++ or Java, and converted into code which the computer understands.

A high-level language is one that people find relatively easy to understand. The code written at this stage is called **source code**.

The machine can understand **machine code**, a translation of the source code into the binary instructions which the machine understands. This code, because it can be executed by a computer, is also known as **executable code**.

The translation, from source code into machine code, is carried out by a program called a **compiler**. The resultant machine code is **portable** to machines of the same type running under the same kind of operating system.

Compiling and debugging large programs can take a lot of time.

Another form of translation that can reduce development time is an **interpreter**. When an error is encountered, the interpreter immediately feeds back information on the type of error and stops interpreting the code. This allows the programmer to see instantly the nature of the error and where it has occurred. He or she can then make the necessary changes to the source code and have it re-interpreted.

As the interpreter is also executing each line of code at a time the programmer is able to see the results of their programs immediately which can also help with debugging.

Sometimes problems that arise at the implementation stage will call for a return to an earlier stage of the development process. For example, it might turn out that an algorithm runs too slowly to be useful and that the designer will have to develop a faster algorithm. Or it might be that the slowness of operation is due not to a poor algorithm but to the hardware capability. In such a case, the development team might have to return to the analysis stage and reconsider the hardware specification.

At the end of the implementation stage a structured listing is produced, complete with internal documentation. This will be checked against the design and against the original specification, to ensure that the project is remains on target.

There are many programming languages used for the implementation of software developments

2.6.1 Review questions

Q21: One of the qualities of a good human computer interface is?

a) It should be Windows-based
b) It can make a program execute faster
c) It can make running a program a less irritable experience
d) It should have lots of colour

Q22: Which one of the following elements is **not** part of software design?

a) The interface
b) The requirements
c) The data structures
d) The main program

Q23: Which one of the following languages is not high level?

a) Visual Basic
b) Pascal
c) Assembler
d) C++

Q24: When high level language code is compiled the code produced is called (choose one)?

a) Executable
b) Object
c) Machine
d) All of these

Q25: Data processed by computers must be structured in some way. Which one of the following reasons is true?

a) To allow for faster processing
b) So that databases can be used
c) Since there might not be sufficient external storage space
d) To cater for different data structures

2.7 Testing in closer detail

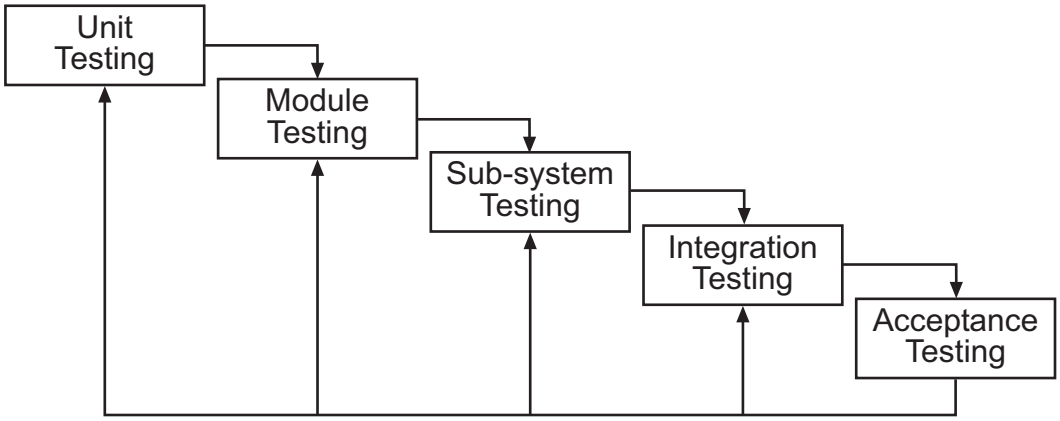
Testing has several purposes. It should check that:

- the software meets the specification
- it is robust
- it is reliable.

Commercial software is *not* exhaustively tested at the testing stage. Software can be complicated and the available time is limited. There must be a balance between creating a product for the market and exhaustive testing. If errors become apparent after release, the company will fix them and release with an updated version.

Test data preparation

Testing can never show that a program is correct. Even with extensive testing, it is almost certain that undetected errors exist. Testing can only demonstrate the presence of errors, it cannot demonstrate their absence. A program can be regarded as succeeding if it passes a test; the test can be regarded as succeeding if it makes the program fail.



One thing you have to watch for is that errors may not cause immediate failure or obvious corruption of your program. Instead they may result in an incorrect output at some later stage.

To detect these errors the running of the program must be traced as the faulty output is only a symptom of the problem rather than the problem itself. The easiest way to trace the program execution is to add "print" statements to the program at key points or use a **trace facility**.

Finally, and most importantly, tests should be devised against the specification, so that you can see whether or not the program does what it is supposed to do.

In the full software development process, now is the point at which test data should be prepared. You do it *before* the coding. That is, before you have invested time and effort in writing the code.

All too often there is a temptation especially at classroom level to start coding a program as soon as possible without having produced adequate test data.

But experience shows that this is a mistake. Once you have written the code you will tend to go easy on it, and let the program's behaviour shape what you expect of it. You are going to be too kind to it. What you see becomes what you expect and what any 'reasonable person'

MURPHY's LAW 1	**MURPHY's LAW 2**
The quicker program coding is started the longer the project will take	Murphy's Law 1 is correct!

Testing follows a test plan or strategy. With most software projects, the usual strategy is to test the software twice. The methods are called:

- **alpha testing** where the software is tested by clients *within* the organisation
- **beta testing** where the software is tested by personnel *outside* the organisation or by certain members of the public. This is sometimes called **acceptance testing**.

Matching definitions - Testing

On the Web is a interactivity. You should now complete this task.

2.7.1 Alpha testing

Test data is based on the specification. Data will be designed to test three aspects of the program:

- **normal operation**: data that the program has essentially been built to process; all outputs should be satisfactory.
- **boundary testing**: data to test that the program functions properly with data at the extremes of its operation; for example, if a number entered is meant to be limited, the program's performance is tested just within the limit, on the limit, and just beyond the limit; as another example, if a table is supposed to have a maximum number of elements, the program is tested to see if it can cope with exactly the maximum and if it can cope when an attempt is made to exceed the maximum.
- **exceptions testing**: data that lie beyond the extremes of the program's normal operation; these should include a selection of what might be called silly data, to test the program's robustness, that a user might enter in a moment of confusion or mischief.

The HCI should be tested in respect of the operations a user might choose. Whatever the user does, the program should return to an active form. The HCI should be tested in two respects:

- Normal user activity: tests should be designed to answer such questions as do all the icons work, are the forms properly connected, are the forms consistent.
- **Unusual user activity**: the HCI should be tested for unusual or potentially disastrous sequences of actions; for example, users get impatient and select things before the program's ready, or select something over and over again because the program doesn't react quickly enough; the program should cope if the user attempts to quit half way through a file operation; and so on.

The program tester draws up a **test log**. This consists, essentially, of four columns: input, reason (for choosing that input), expected output (expectations being formed on the basis of the specification), and actual output. Naturally, the first three columns are filled in before testing and the fourth during testing.

When a program is highly interactive, people often talk of its behaviour, and the test log would record input, reason, expected behaviour, and actual behaviour.

Faults that become evident during testing are known as **bugs**. If bugs are identified, the program is sent back, with the test logs, to the programming team for **debugging**. This process is likely to be iterative: testing, finds bugs, they get fixed, the program's tested again, more bugs are found, and so on.

It may be that bugs reveal flaws that were introduced at an earlier stage of the process, at the design or even at the analysis stage. If this is the case, the documentation for each stage of the development process will need to be corrected.

A standard technique to identify potential errors is to conduct a **dry run**. This involves taking test data and a listing of the relevant part of the code, and calculating exactly what would happen to the data if it were to pass through that code. It is a pencil and paper exercise.

2.7.2 Beta testing

Otherwise known as acceptance testing it takes place after alpha testing. The idea is to subject a completed program to testing under actual working conditions.

If a program has been developed for use by particular clients, it is installed on their site. The clients use the program for a given period and then report back to the development team. The process might be iterative, with the development team making adjustments to the software. When the clients regard the program's operation as acceptable, the testing stage is complete.

If a program is being developed by a software house for sale to a market rather than an individual client, the developers will provide an alpha-tested version to a select group of expert users such as computer journalists and authors, and also makers of related computing products such as printers.

This is of benefit to both parties: the software house gets its product tested by people who are good at noticing faults, and the writers get to know about products in advance; which further benefits both parties when the final production software is released, the software house getting publicity and the writers receive credit for being up to date.

People involved in beta testing will send back error reports to the development team. An error report is about a malfunction of the program and should contain details of the circumstances which lead to in the malfunction. These error reports are used by the development team to find and correct the problem.

2.7.3 Review questions

Q26: Which one of the following statements is true relating to alpha testing?

a) Testing is done by the users
b) Testing is done within the organisation
c) Testing is done by specialist companies
d) Testing is done by the client

Q27: The statements refer to a dry run testing of software. Which one is involved in the process?

a) No software is involved
b) No users are involved
c) No computers are involved
d) No programmers are involved

Q28: Beta testing differs from alpha testing in one of the statements below. Which one?

a) The program is tested by the clients
b) The testing is more rigorous than alpha
c) The testing is for market research
d) The program is tested by specialist companies

Q29: Testing can never show that a program is correct. Which option best describes why?

a) It is far too exhausting
b) You cannot account for all possible errors
c) It is too time consuming
d) Perfect programs do not exist

Q30: During alpha testing, the program is usually subjected to **exceptions testing**. This means:

a) The input of silly data
b) The input of large numbers
c) The input of small numbers
d) All of these

2.8 Documentation in closer detail

Documentation is intended to describe a system and make it more easily understood. Documentation will consist of:

1. user guide
2. technical guide

Some information may appear in both guides; eg system specification.

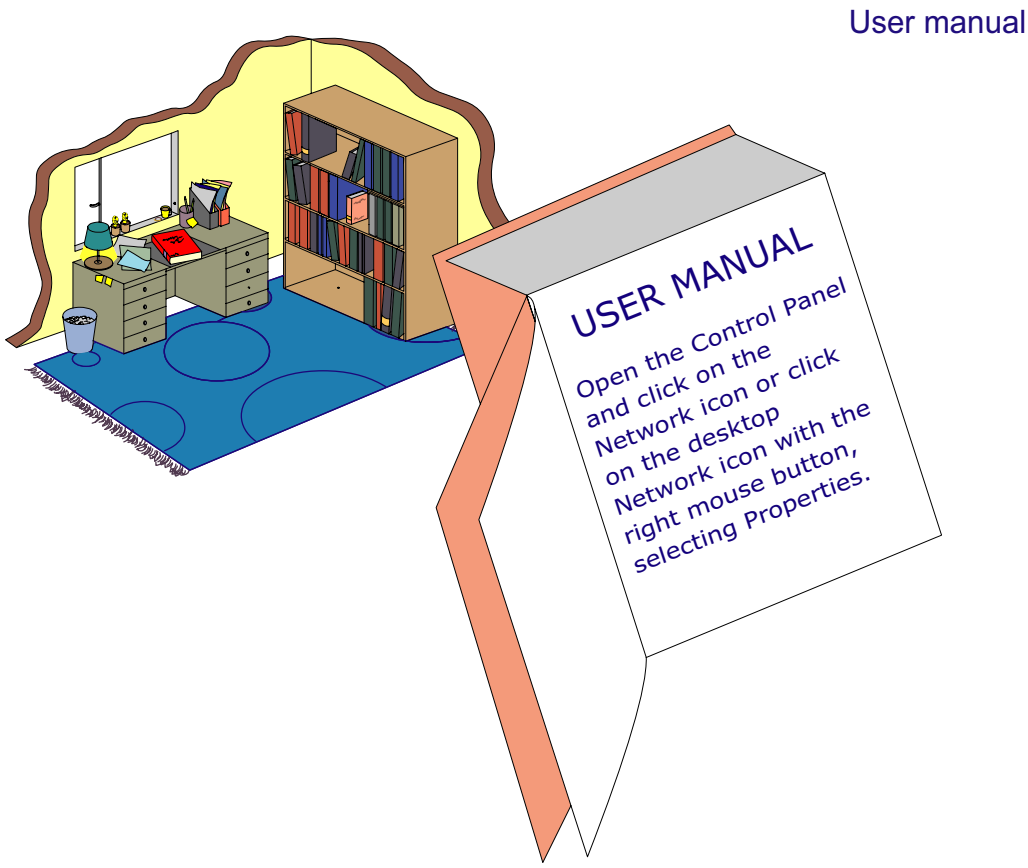
Internal documentation such as remarks or comments in the code are for the benefit of the development team. It will help if changes have to be made to the software in the future.

Other documentation for the development team includes all the documents produced in the software development process: requirements specification, program design documents (for the HCI and for the structure and logic of the underlying code), a structured listing of the code, and a test history.

2.8.1 User guide

The user guide contains information about how to install, start and use software. It should also contain a list of commands and how to use them. Where there is a significant HCI, the guide will show each form, menu, and icon, and associated instructions about their use.

User guides may be supplied as paper manuals, often with separate manuals for installation, getting started and the user guide.



Increasingly software vendors supply the manuals on disc where they may be available in (pdf) or hypertext markup language (HTML).

Paper manuals are costly to reproduce; manufacturers frequently include electronic files which provide up-to-date amendments. This lets the user read up to date information which could not be included in the original paper manual.

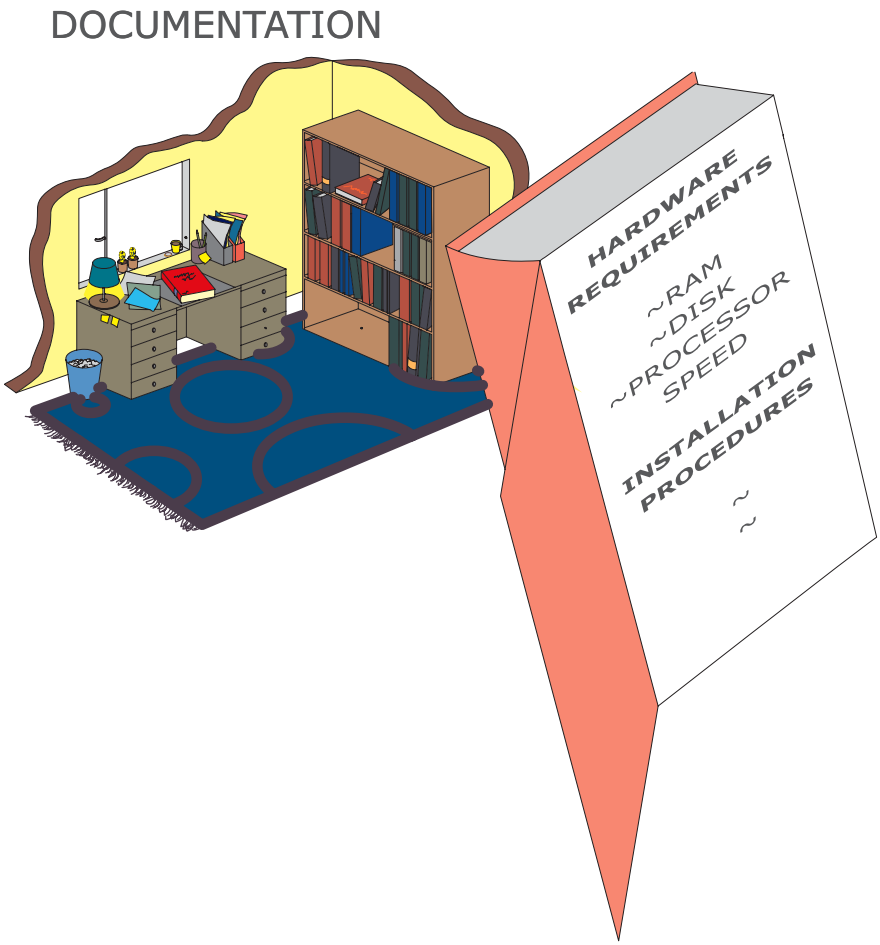
It is common for sample files to be included which complement the tutorial and provide the user with demonstration files.

The program should contain a help facility. It is common for on-line help to be presented in three tagged pages: Contents, Index, and Search. The contents present the help chapter by chapter; the index refers to certain key words in the chapters; and search offers the facility to locate key terms within the guide.

Onscreen help may be provided in Forms which have hints programmed into them, so that if the cursor lingers on an icon or some other significant part of the form, a message will be displayed explaining the icon's purpose or the significance of that part of the form.

2.8.2 Technical guide

The technical guide will contain information about the hardware and software requirements of the program. The hardware specification will include details of the processor type and speed, RAM required, RAM desired, monitor resolution, graphics and sound card specifications etc. It will also contain instructions about configuring the program.



Software designed to operate, or run on networks can be very complicated and require a good deal of expertise. Technical guides can be very large and cumbersome and difficult to navigate.

2.9 Evaluation in more detail

The evaluation is important for the user and the software author. There are two reasons for conducting an evaluation:

- does the software meet the users requirements?
- how can the software house improve the SDP in future projects?

The performance of the system can be assessed in various ways:

- how closely does it fit the system design?
- how well does it answer the problem specification?

Questions may also be asked about matters such as:

- was the project within budget?

- was the project completed to schedule?

The development team will wish to review the project, perhaps to learn from any mistakes to ensure they incorporate good points in future programmes.

Software houses aspire to produce new and better versions of their software. They will study press reviews and note any contents and criticisms. New or forthcoming changes (in technology, in operating environment, and so on) are also taken into account. When the evaluation is complete, work begins on the next version of the system.

2.10 Maintenance in closer detail

Once the software is operating, the users will need support. In the case of a bespoke system, the development team (or the organisation it works for) will offer training in the use of the new system.

Creators of software systems often establish help desks, so users can obtain advice about the software.

Software does not wear out, in any physical sense, but the presence of errors or omissions will give rise to the need for **maintenance**.

Software maintenance always involves a change in the software with the accompanying probability that additional errors may be introduced. It is essential to ensure that adequate quality control is in place.

Maintenance documentation is designed for computer professionals rather than users, and includes the documentation produced by the development process: systems documentation, such as the specification and the design documents and program

documentation, such as the structured listing.

There are three types of software maintenance:

- corrective
- adaptive
- perfective

Corrective maintenance is concerned with errors that escaped detection during testing but which occur during actual use of the program. Users are encouraged to complete an error report, stating the inputs that seemed to provoke the problem and any messages that the program might have displayed. These error reports are invaluable to the development team, who will attempt to replicate the errors and provide a improved solution.

Adaptive maintenance is necessary when the program's environment changes. It allows the authors to provide a program which responds to changes in the operating environment. For example, a change of operating system could require changes in the program, or a new printer might call for a new printer driver to be added to the program. A change of computer system will require the program to be ported to the new system.

Perfective maintenance occurs in response to requests from the user to enhance the performance of the program. This may be due to changes in the requirements or new legislation. Such maintenance can involve revision of the entire system and can be expensive.

Matching definitions - Maintenance

On the Web is a interactivity. You should now complete this task.

2.11 Weaknesses of the Waterfall Model

Problems are usually encountered when you use the waterfall model:

1. Real projects rarely follow a linear, sequential flow. Apart from any software problems, people change their minds, and often there may be changes in legislation which mean that the program must be altered in order to comply with the new regulations. No matter what the reason, iteration always occurs and this creates problems because much of your work has to be re-examined and revised;
2. It is difficult for the customer to state all requirements explicitly at the start of developments. The waterfall model depends on this and has difficulty incorporating customer uncertainty;
3. Clients are frequently excluded from the development. The working version of the program will not be available for the customer to see until late in the development cycle;
4. Developers work in isolation from the clients, often for months, only for the clients

to be disappointed with the results. Many developers value feedback from clients as the project progresses;

5. Errors arising from incorrect requirements will not be obvious until late in the cycle, by which time they will be difficult and expensive to fix. There is nothing so depressing as delivering months, sometimes years, of work to the customer only to be greeted with the response, "That's not what I wanted at all."
6. The waterfall model does not address project management or software maintenance.

There are alternative models that have been developed for two reasons:

- perceived shortcomings in the traditional approach;
- advances in hardware and software.

2.12 Alternative models

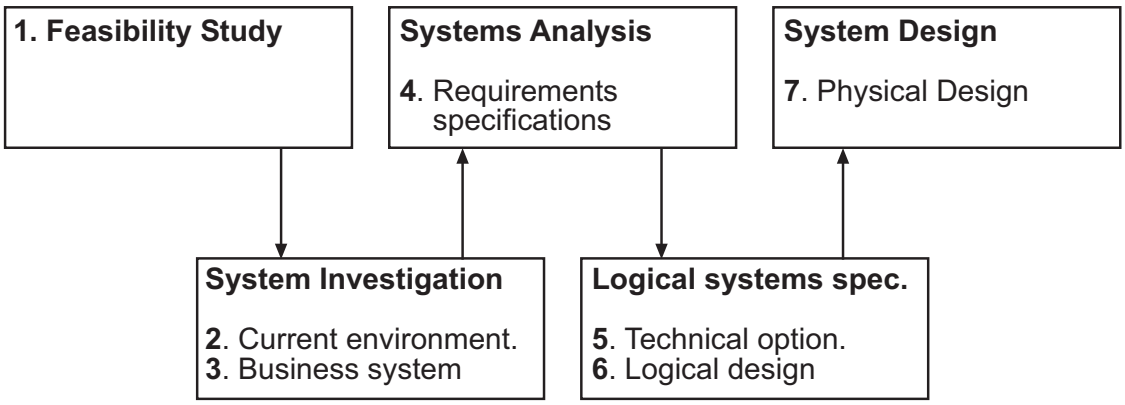
The traditional model remains the basis for these alternative models and their subsequent refinements. There are many models. We will briefly describe the following:

- structured systems analysis and design (SSADM)
- prototyping
- rapid application development (RAD)
- commercial *off-the-shelf* software (COTS)
- spiral
- open source

2.12.1 SSADM

The Structured Systems Analysis and Design Method (SSADM), concentrates on the analysis and design aspects. It deals with only part of the overall process and leaves the other aspects to other specialist models. The stages addressed by SSADM are: analysis, requirements specification, logical design, physical design and program specification.

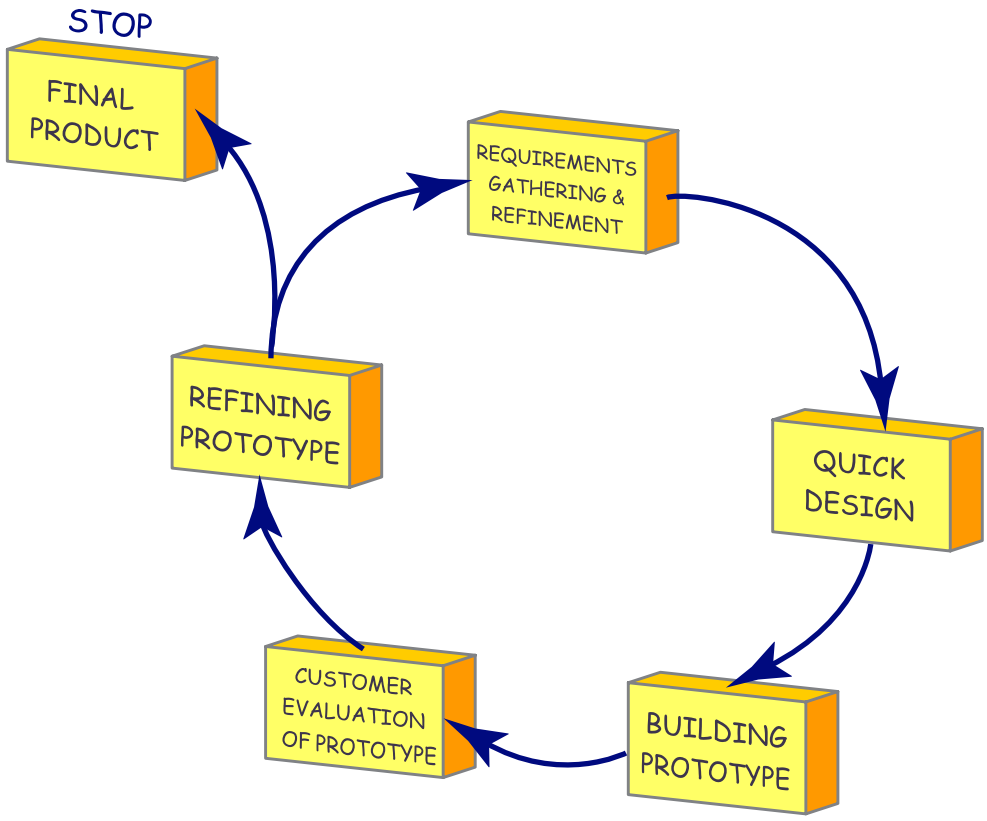
The following diagram outlines the typical structure of SSADM:



This model was developed by Learmonte and Burchett and is widely used in the UK by government and public organisations where it first appeared at the Central Computer and Telecommunications Agency (CCTA). It is frequently used in conjunction with a planning process model; *eg PROMPT II, a project management methodology* where it is now regarded as a *de facto* standard.

2.12.2 Prototyping

A prototype is a small scale representation of a proposed design. In prototyping the primary focus is the interface. The interface can be put together quickly and shown to the clients. Prototyping involves clients in the software development process. The clients can make suggestions which can, in turn, be incorporated into a revised version of the prototype. As such it involves feedback and is said to be iterative. This iterative process can be repeated to make further refinements. When the prototype has been suitably refined and agreed, work on the full project will begin.



Prototyping had its proponents in the old days of text based screens, but it has further developed since the appearance of the Graphical User Interface (GUI). Programming languages such as Delphi, Visual Basic and Kylix make it easy to construct graphical user interfaces.

Amendments suggested by the clients are easier to incorporate at the prototype stage rather than later. When the prototype has been amended, it is again submitted to the clients. The process continues until the clients are satisfied with the interface. When the HCI has been agreed, work starts on the detailed processing.

Disadvantage of prototyping

Users often perceive the interface to be the program. Clients, having seen a prototype one week, find it hard to understand why the entire program is not ready the following week. Prototyping, at least in respect of designing the HCI, is still welcomed by the clients and programmers.

2.12.3 Rapid Application Development (RAD)

RAD requires languages like Delphi or Visual Basic. These are **event driven** languages which allow easy construction of interactive forms.

Another term for this sort of approach, which stresses the partnership of clients and developers in the production of the final software, is Joint Application Development

The term RAD has not been current for very long but has already evolved and its meaning altered. Originally, RAD was applied to small but highly interactive programs that could be completed in a day or two. The programmer would simply study the specification, plan the windows needed and how they would link together, and create

the program. This could be used as a prototype: the client could look at it, even take it away and test it; amendments could be made easily. Many programmers wanted to extend this approach to deal with large programs.

Programmers use it for larger projects and involve their clients all the way through. The process is iterative and makes use of feedback from clients. The advantages are that projects tend to be finished quickly; clients feel involved and time isn't wasted creating things that the clients don't want.

The process has been formalised and centres on a prototyping loop of design, construction (of prototype), implementation (use of prototype) and analysis (of feedback from the prototype). Note that implementation has here a different meaning from its meaning in structured design. Results of the analysis (suggestions for changes in design or alterations to requirements) are then fed back into the design stage, and the loop runs again. When the prototype seems satisfactory, the process moves on to full implementation.

Disadvantages of RAD

A disadvantage of RAD is that the resulting software can be patchy, representing a series of ad-hoc responses on the programmer's part than a methodical attempt to provide a robust system. Problems with such software, which might arise at the testing or maintenance stages, can be difficult to debug.

2.12.4 Commercial Off The Shelf (COTS)

When structured design was invented there was not, by today's standards, a lot of software available. Software houses create software packages for use by individuals and organisations. Rather than have programmers develop a new system from scratch, many organisations first look to see if a packaged software solution already exists, for sale or for hire. This approach is sometimes called the **COTS** approach.

COTS packages can be very expensive. Commercial software is generic and requires to be configured to the client's requirements. The client's existing software may need to be adjusted to suit the new package. Furthermore, the package is unlikely to fulfil all the organisations requirements and additional software may need to be written to meet these requirements.

2.12.5 Spiral model

This model was introduced by Barry Boehm in 1988. It is a risk driven approach rather than code driven. It incorporates management and risk assessment components. The development of a final product is based upon successive refinements.

2.12.6 Open Source

All the models described above are used to represent and enhance the production of commercial software. As such it is envisaged that the client has a specific goal, the company wishes to sell a product and make a profit and that programmers are employed to generate this software.

There is an alternative philosophy based on the Free Software Federation. The philosophy is based on the following:

- The proliferation of the Internet and the increase in ease of communication has allowed geographically distant programmers to collaborate in ways not previously possible. This in turn has allowed an increase in the development of many open source projects
- The open software movement represents a growing body of users and programmers who have a totally different philosophy to the development and ownership of software. Discussion of the open source movement is contained in the article *The Cathedral and the Bazaar by Eric S Raymond*
- A considerable amount of software has successfully been created using this Open Source model; *eg examination of the Netcraft web site will illustrate the number of web servers globally which are operating using Apache.*

2.13 Summary

The following summary points are related to the learning objectives in the topic introduction:

- understand the mechanism of the software development process;
- the process is iterative and involves a continual revision and evaluation at each phase of the process;
- it is a time-consuming procedure from initial specification to a working program;
- various models exist to aid the software development process but none are perfect.

2.14 End of topic test

An online assessment is provided to help you review this topic.

Topic 3

Design notation, data flow and evaluation

Contents

Prerequisite knowledge

Before studying this topic you should be able to:

- *describe and use pseudocode;*
- *describe and use a graphical design notation structure diagram or other suitable method;*

Learning Objectives

- *understand the nature of graphical design notations*
- *be able to use a graphical design construct*
- *understand the nature of pseudo-code*
- *be able to use pseudo-code*
- *understand the nature of top down design and stepwise refinement*
- *understand the need for systematic and comprehensive testing*
- *understand the terms robustness, reliability, portability, efficiency and maintainability in the context of software evaluation*

Revision

Q1: At the design stage of the software development process pseudocode may be used to represent a solution to a problem. Which of the following best describes the use of pseudocode?

a) It uses ordinary English words
b) It is high level language dependant
c) It is very useful in complex program designs
d) It mostly uses high level language key words

Q2: A structure diagram is a valuable aid to the programming team. This is because:

a) They are easy to use and look good
b) They are well-structured so allow for faster program execution
c) They are the best used with programs that produce a lot of output
d) They can reduce the overall development time of a program

Q3: Pseudocode can be considered to be an intermediate stage between:

a) High level language code and machine code
b) English and high level language code
c) English and machine code
d) None of these

Q4: At what stage of the software development process could a running program be modified or updated to a newer version?

a) Testing
b) Evaluation
c) Maintenance
d) Implementation

Q5: A computer program is designed to accept input values between 0 and 99 as whole numbers. If the value 56.8 was entered this would be an example of:

a) An input error
b) An exceptional input
c) Unacceptable output
d) All of the above

3.1 Introduction

In this topic you will learn about the various methods that are used to aid the system developers and programming team to implement solutions according to program specifications. The first important issue you will come across is that program coding is only attempted after extensive and rigorous series of planning and design stages are completed beforehand. Efforts at these initial stages pay dividends at the coding stage in terms of testing, debugging and maintenance of the programs.

3.2 Tools and Techniques

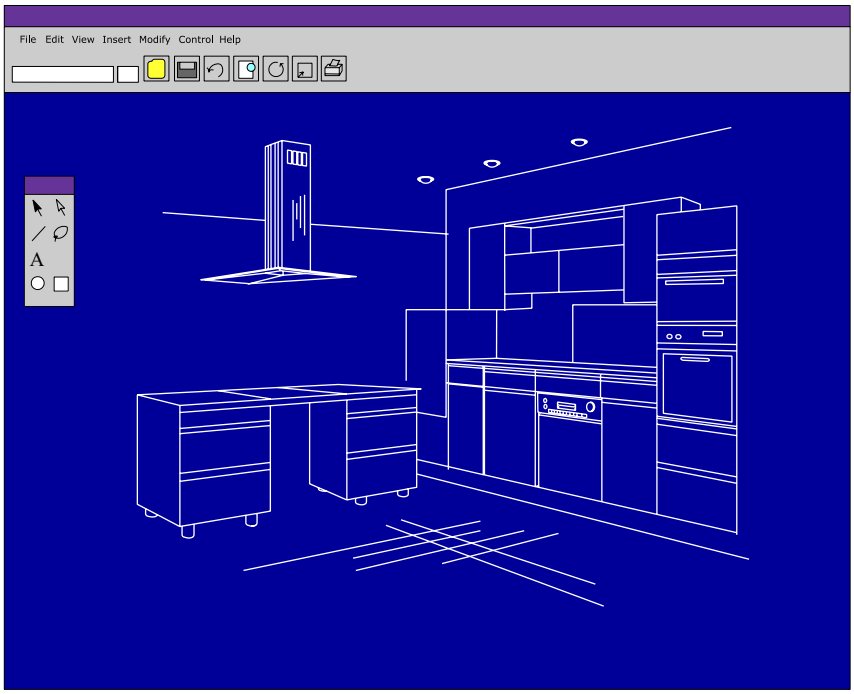
This section describes the tools and techniques used in the software development process. Initially good program design will save much cost and time in the later stages of development such as testing, debugging and maintenance. You will see more of this later.

The main tools and techniques include:

- Design methodologies
- Test data
- Structured program listing
- Comprehensive error reporting
- Module libraries

3.3 Design methodologies

The design of software is something of an art and normally follows a clear design **methodology**. A methodology is an agreed technique used to design software. It includes both approaches to designing software and the notations used to represent the design.



Design documentation will often include more than one notation, eg structure charts for the higher levels, to provide an overall picture, and pseudo-code for the detail. Often the design notations are used at different stages in the design process and serve to provide information to different audiences.

3.3.1 Top-down design with stepwise refinement

Stepwise refinement is associated with Nicholas Wirth the creator of the Pascal and Modula languages. It is an approach used throughout computing and in many other fields as well. The idea behind it is this:

1. a problem might be difficult to solve as it stands. So you try to break it down into a set of smaller problems which might be solved more easily. This represents the first step of the process
2. you then take the smaller problems, one at a time, and break them down into still smaller problems
3. you keep repeating this process of breaking the problems down until all the problems facing you are small enough to handle

At this point you are able to create detailed design specifications, which can be turned into programming code.

This notion of breaking larger, difficult problems down into smaller, easier-to-solve ones is the refinement. Stepwise refinement is sometimes called top-down design, for the obvious reason that you start the process at the top, with the problem as a whole, and work downwards. This is shown in Figure 3.1

One other advantage of this systematic approach is that it also automatically gives a structure to your solution.

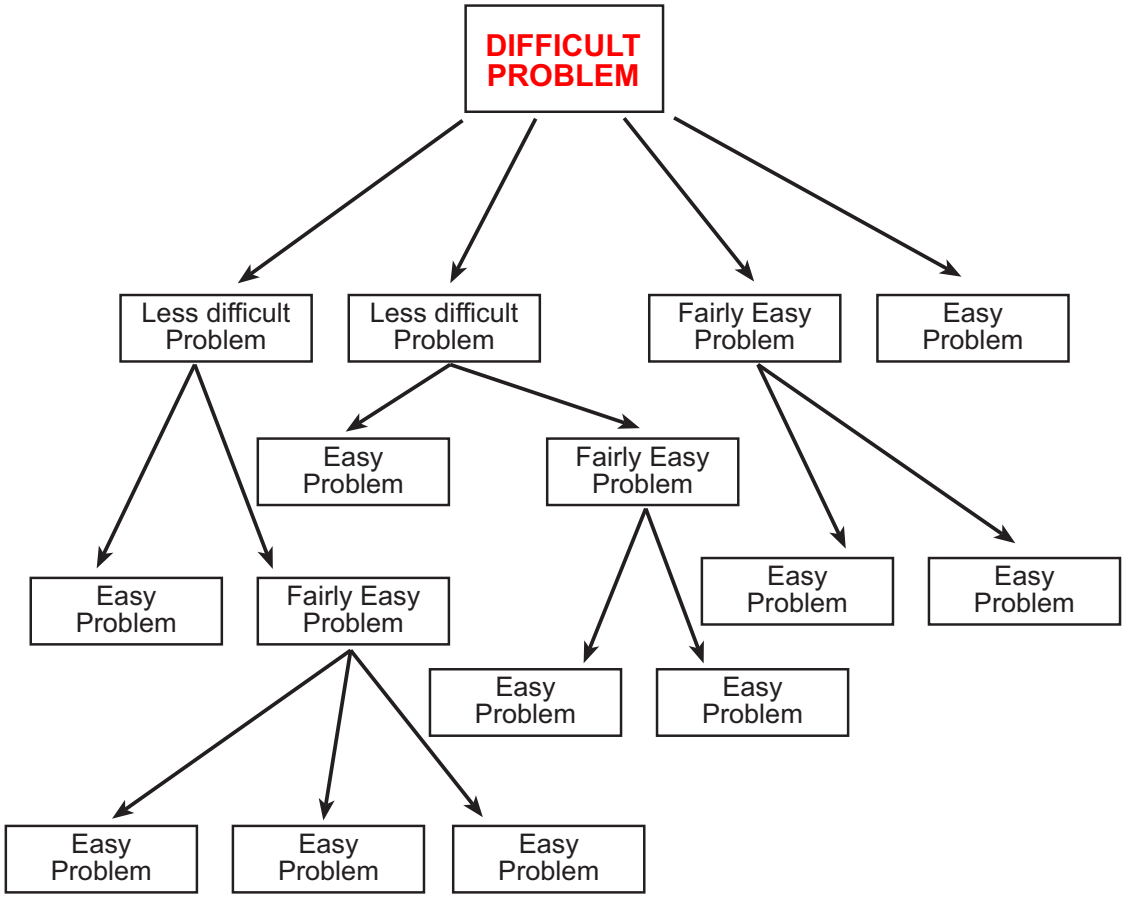


Figure 3.1: Stepwise refinement

Stepwise refinement allows the designer to concentrate on one small part of the problem at a time. If thinking about the problem as a whole, the designer does not have to bother with details. In thinking about a portion of the detailed design, the designer does not have to bear in mind all the rest of the problem. This makes the process manageable.

Once manageable parts have been identified the analyst can assign individual tasks to different teams of programmers. The complex task is made more straightforward by the system of *'divide and conquer'*.

Programmers often use **stubs** when using a top down technique. A stub is an outline of a module, that does little more, at run time, than declare its presence or return a value of appropriate type. When all the stubs are in place, the program as it stands can be tested to make sure that all the stubs are properly linked. Then the detailed work can begin.

It may have occurred to you that if top-down design exists then what about bottom-up analysis?

Bottom-up design begins with the lower levels of detail, for example a device driver for a peripheral that will require to be written around the program codes needed to operate the device. The emphasis here is to write the individual modules, and knit them all together to form the final program. With *object-oriented* programming the bottom-up approach has been partially revived.

3.3.2 Structure charts

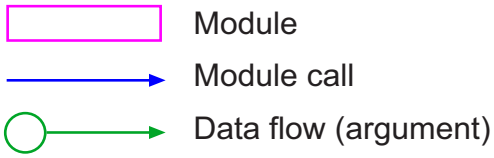
An important approach in practically all methods of analysis is to draw a picture of some kind. Analysts use pictures to present a synopsis of a system which shows:

- the main elements of the solution
- how they fit together.

It is not enough for the analyst to have a design in mind. It must be represented in a form that can be used by all members of the team. One method is to use structure charts, which are sometimes called structure diagrams. A structure chart gives a picture of the design.

Structure charts depict programs or parts of programs. Styles vary in detail, but in essence structure charts consist of boxes, lines, arrows and text. A box represents a block of code and has a name. Usually, it is a descriptive expression starting with a verb; it shows the block's purpose.

Structure charts can be drawn according to various sets of guidelines. These are the basic graphical elements:



A structure chart for a program that simply gets in some data, changes it in some way and produces output might look like Figure 3.2:

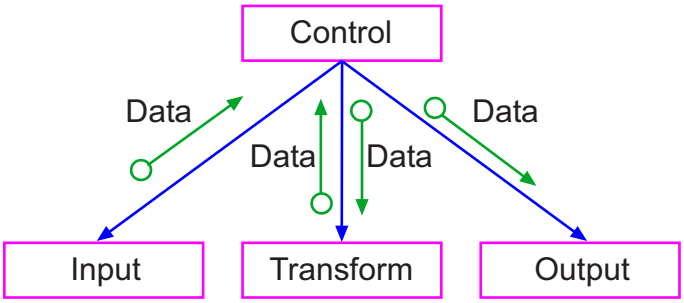


Figure 3.2: Software Design Structure

Structure charts are read from top to bottom. The line joining two boxes indicates that the lower is called, or brought into action, by the upper. The lower box represents one of the things the upper box has to do. Along each line, names and arrows describe the data that flow from one block to another and the direction in which they flow.

Each block in the structure chart represents a section of code to be written. These sections have the generic name module.

The modules in a structure chart will become modules of code in the finished program. The designer must give the modules meaningful names (rather than Block 1, Block 2 etc.) which describe what the module does. This makes the design easier to follow.

Structure charts emphasise modular design as they are hierarchical. The project as a whole is at the top of the hierarchy. A structure chart shows the relationship between modules and in particular shows which modules contain calls to modules lower down in the structure. A complete structure chart of a large software system shows the relationship between all the modules in the system. A structure chart conveys selected information with clarity, rather than presenting all the information at once.

By convention, structure charts are kept simple, with no more than half a dozen blocks to each chart. What matters is that the picture should be clear and easy to bear in mind. If a block needs further refinement, that is represented in another chart.

Describing a structure chart

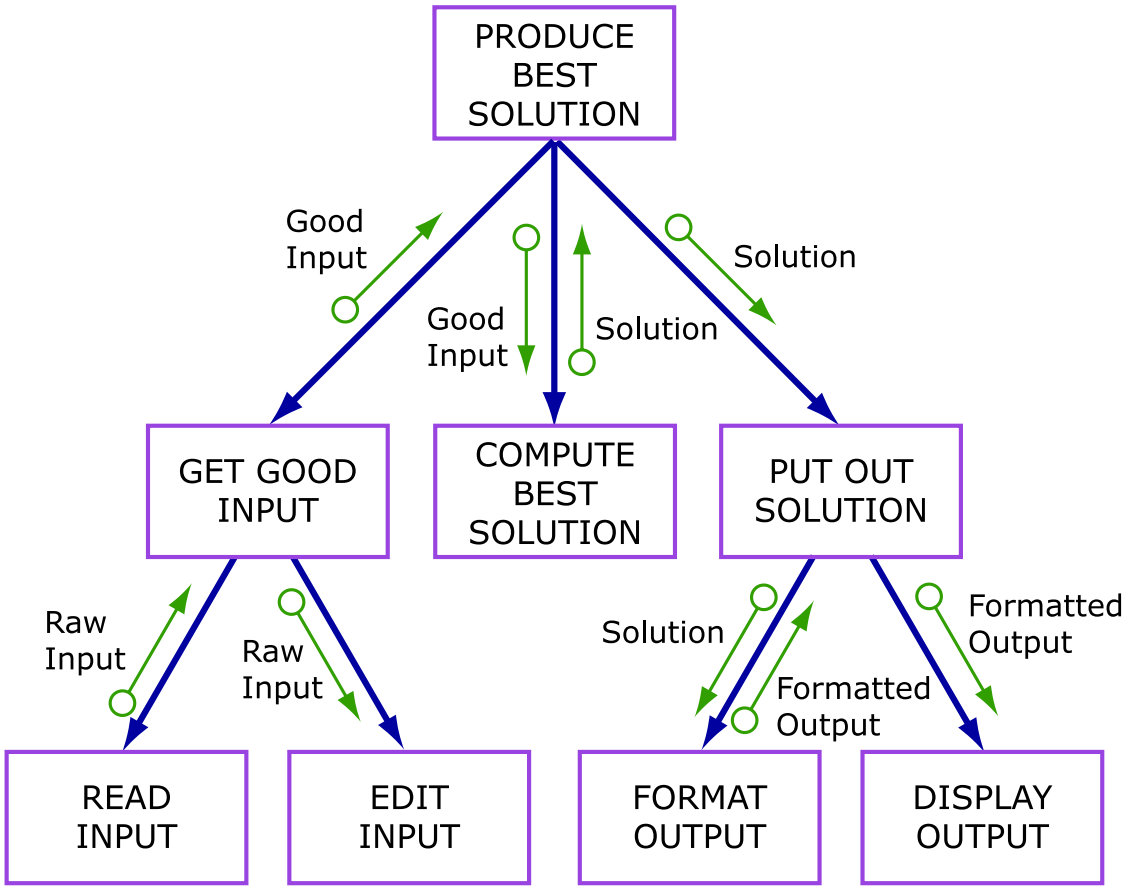


Figure 3.3: Sample structure chart

Figure 3.3 shows a structure chart. Make a list of all the modules and all the data flows. Remember to make a sensible guess at what each block is supposed to do - it's name should be a good indication of this!

What sort of program do you think is represented by the whole structure chart?

3.3.3 Jackson Structured Programming

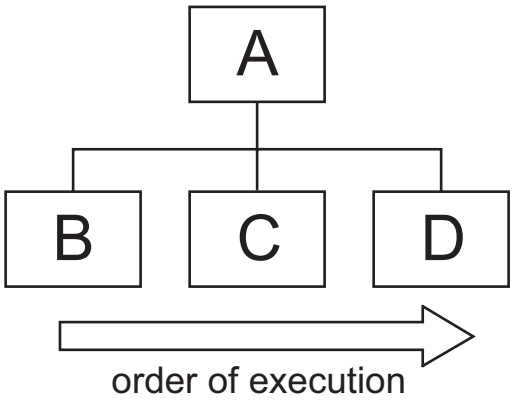
Other methods for structured programming were developed. One, explicitly for use on small projects, was developed by a man called Michael Jackson in the 1970's and his

book on the subject came out in 1975. His method became known, fairly enough, as **Jackson Structured Programming** (JSP).

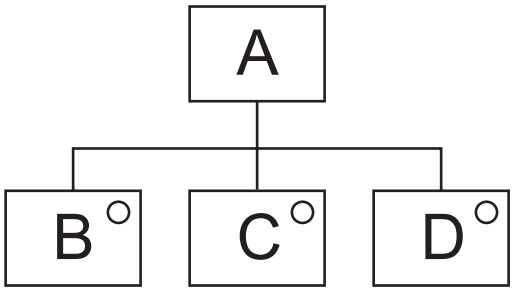
JSP is designed for use with relatively small programs, for program design in the small. It uses only the three control structures necessary for structured programming: sequence, selection, and iteration.

The design is made up of boxes or nodes, each of which represents a part of the process. Boxes have names.

A plain box represents action of some kind. This is sometimes called a normal node. Boxes are always read - and, in the end, executed - from left to right:

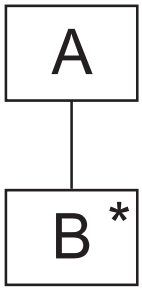


A box with a circle in it is a selection node. This means that it is part of a selection sequence. It does not necessarily mean that it is a node that actually does any selection itself.



In this diagram, '**a**' is the node where the selection takes place. As a result of the selection, '**b**', '**c**' or '**d**' happens. Note that '**b**', '**c**' and '**d**' must all be selection nodes: if one child of a node is a selection node, they must all be. The right-most node, in this case '**d**', will contain default behaviour.

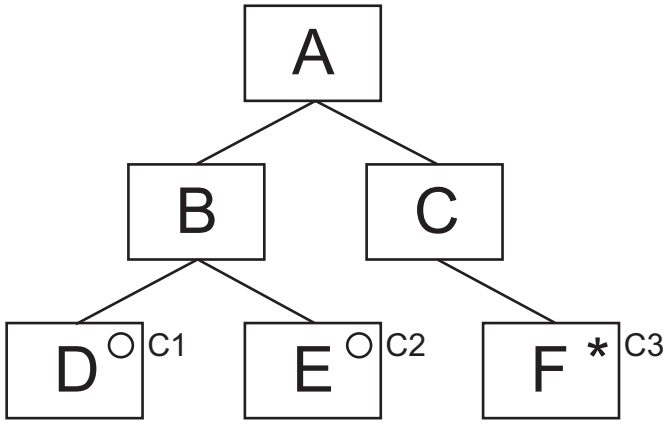
A box with a star or asterisk in it is an iteration node. The code it represents must be repeated a certain number of times.



To understand what a node is doing, you need to investigate the next level down.

Many people feel this is a bad way of working and that some sort of sign should be present to indicate that a node involves, rather than is involved in, selection or repetition. In fact, some designers use these signs to do that.

Conditions are indicated by the letter C and a number alongside the box. The number is used to distinguish between the different conditions that might be involved in a program. The conditions in turn need to be spelled out somewhere on the diagram.



Each box represents a block of code, either a procedure or a function.

When the JSP diagram is complete, the corresponding pseudo-code is then written for each box.

Characteristic of good software design

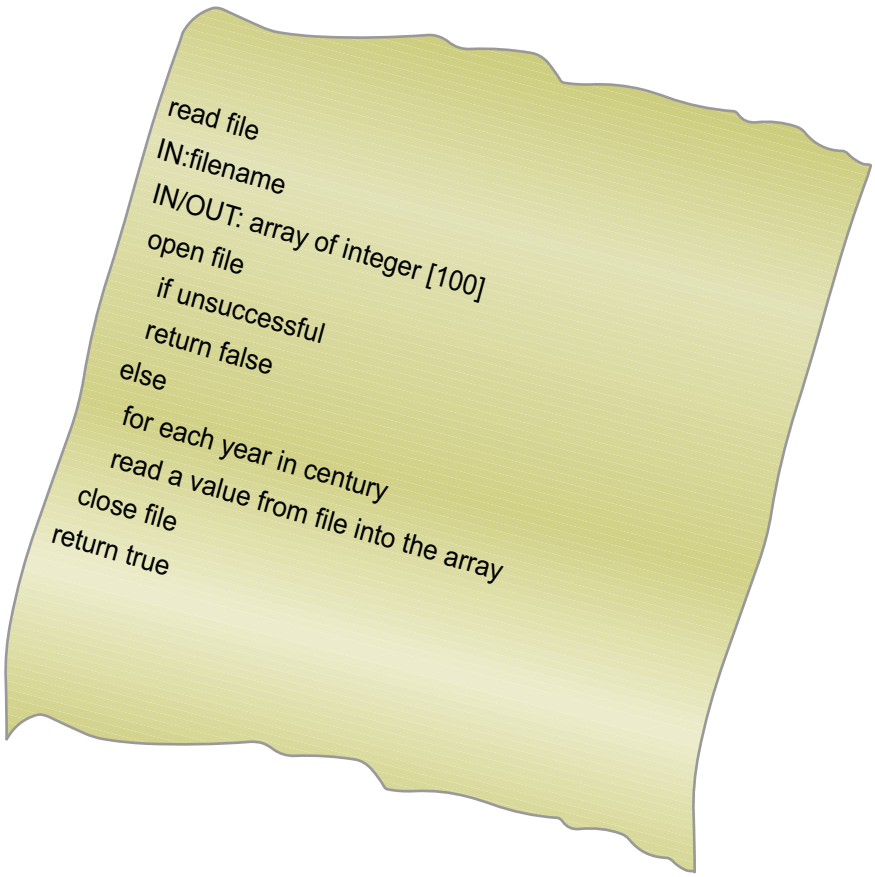
On the Web is a interactivity. You should now complete this task.

Structured design

On the Web is a interactivity. You should now complete this task.

3.3.4 Pseudo-code

Another method is to describe the design in terms of pseudo-code. This is mostly used for working out the details of a design. At this level, pictures don't really help. Pseudo-code is a terse way of writing that lies somewhere between code and natural language (such as English). It represents an understanding of the solution that can be turned into code but can still be understood by people. Pseudo-code follows the indenting conventions of the programming language to be used for the project. This indentation makes the pseudo-code easier to follow and to understand.



Pseudo-code is a way of writing about a process without having to bother about details which are simply a matter of coding. So that you could write:

without having to bother with the details of how this is going to be implemented.

Or you could write:

```
convert entry to number
```

without having to remember exactly how this is done in the chosen programming language.

Pseudo-code frees us to get on with thinking about the details of program design, without our having to stop and look things up in books or worry about whether we have got our syntax exactly right. You use pseudo-code to explain what a process will do in a clear and concise way. One of the advantages of pseudo-code is that it can be translated into programming language code fairly easily. Very easily, in fact, if you are using languages

such as C, C++, Pascal, Smalltalk or any of the family of block-structured languages which descended from Algol in the 1960s. Pseudo-code also translates well into Cobol, so by writing out your program in this way, you are opening up the possibility of re-writing it in a large number of different forms, suitable for different machines.

Many designers use a numbering system for parts of the design. Each block in the structure chart has its identifying number and each line of pseudo-code is also numbered. These numbers enable parts of the program to be related, and show their dependencies. For example, a block number 3.1 in a structure chart might indicate that this is the first sub-module in the third module in the main part of the program. A line of pseudo-code 3.1.10 would indicate the tenth line in this module.

Here is an example of a routine that you will meet in your programming exercises:

Counting occurrences

1. set counter to zero

2. prompt user to input search value

3. set pointer to start of list

4. do

4.1 compare search value to item at current list position

4.1.1 if search value = current item then

4.1.2 increment counter by 1

4.1.3 set pointer to next position in the list

4.2 loop until end of list

5. output number of occurrences (counter)

Characteristics of pseudo-code

On the Web is a interactivity. You should now complete this task.

3.3.5 Review questions

Q6: Which one of the following notations is commonly used to describe software logic?

a) Structure charts
b) Stubs
c) Pseudo-code
d) All of the above

Q7: What statement refers to the term **top-down design**?

a) It is a software testing approach
b) It is breaking complex problems down into smaller units
c) It is the detailed design of software logic
d) It is the use of pseudo-code

Q8: The main idea behind structure charts is (choose one)?

a) To allow descriptive expressions of purpose
b) To show the main elements of a solution
c) To show the various elements fit together
d) All of the above

Q9: In the context of pseudo-code, which statement concerning the user is true?

a) Thinks more about the solution to the problem
b) Thinks more about the hardware
c) Thinks more about how the program will run
d) Thinks more the speed of execution of the program

Q10: One disadvantage of NOT using stepwise refinement at the design stage is that it:

a) Reduces the chance of errors being introduced
b) The designer does not have to bother with too much detail
c) Makes the process more complex to novices
d) The designer can concentrate on a small part of the problem at a time

3.4 Test Data

When you design your program you should include fail safe mechanisms and anticipate user errors. We know this is obvious, but fewer bugs mean that you have to spend less time in fixing your code. Designing your input routines so that they do not crash, means that you do not have to spend time re-writing them afterwards.

Fixing errors can be very expensive because as faults are removed the cost of finding and removing remaining faults increases exponentially, as shown in Figure 3.4 and in Figure 3.5. The last few errors in a program are horrendously expensive to remove - or even to find. You may wonder why this is so, but all the simple and straightforward tests have found all the simple and straightforward errors. What are left are the more complex, intermittent ones which require a very large expenditure of time and effort (and hence money) to fix. Note that the cost scale in Figure 3.5 is logarithmic, not linear. Take a look and see how fast the relative costs soar the further along the project path you are. This is because so much earlier work has to be undone and then re-implemented to fix the errors. This is another good reason for careful analysis and design.

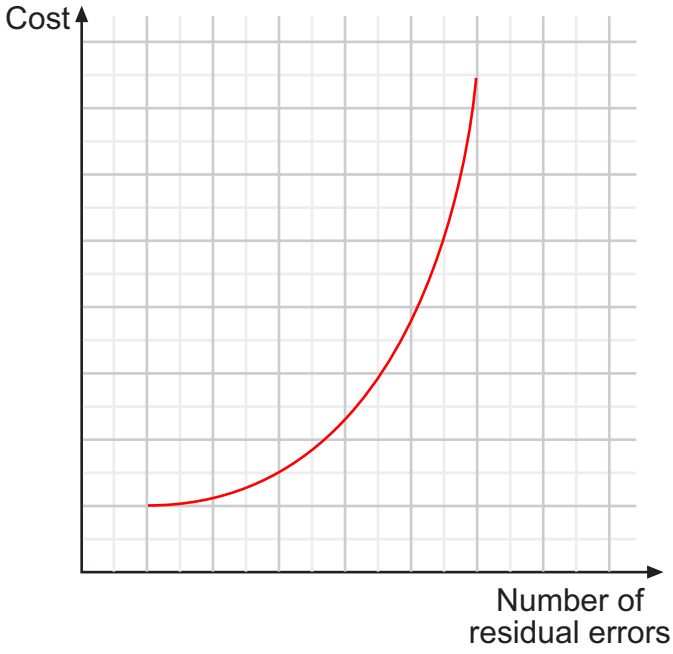


Figure 3.4: Cost of Residual Error Removal

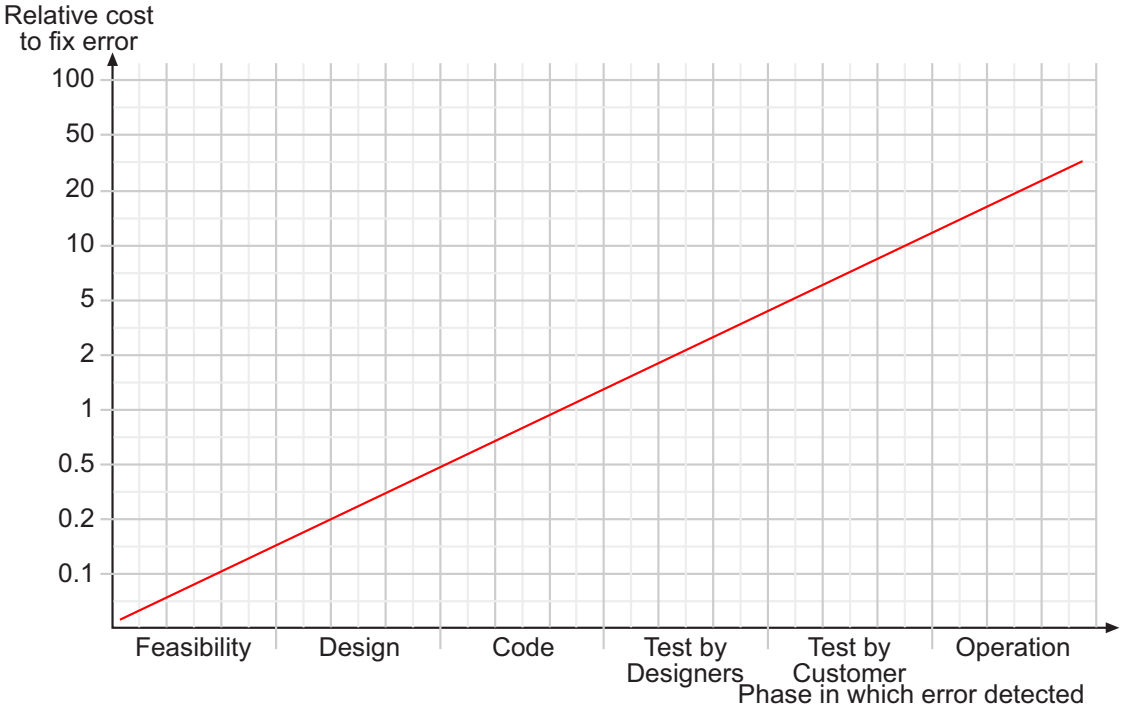


Figure 3.5: Cost of errors in software projects

The purpose of test data is to determine that the system behaves as expected and is correct according to the program specification. Tests may be used for different purposes. These include:

- module testing, on components
- integration testing, components function as a unit

- system testing, black box
- acceptance testing, ensures system is ready for operational use.

3.4.1 Test data preparation

Testing can never show that a program is correct. Even with extensive testing, it is almost certain that undetected errors exist. Testing can only demonstrate the presence of errors, it cannot demonstrate their absence. A program can be regarded as succeeding if it passes a test; the test can be regarded as succeeding if it makes the program fail.

One thing you have to watch for is that errors may not cause immediate failure or obvious corruption of your program. Instead they may result in an incorrect output at some later stage.

To detect these errors the running of the program must be traced as the faulty output is only a symptom of the problem rather than the problem itself. The easiest way to trace the program execution is to add "print" statements to the program at key points. When you write your programs, we are sure you will become very familiar with these!

Finally, and most importantly, tests should be devised against the specification, so that you can see whether or not the program does what it's supposed to do.

In the full software development process, now is the point at which test data should be prepared. You do it *before* the coding. That is, before you have invested time and effort in writing the code.

It is often a temptation to leave making up the test data until you have written the program. But experience shows that this is a mistake. Once you have written the code you will tend to go easy on it, and let the program's behaviour shape what you expect of it. You are going to be too kind to it. What you see becomes what you expect and what any 'reasonable person' would expect.

This is a problem with all programmers but perhaps especially so if you are only beginning programming. You may have only a hazy idea of what you are doing. Getting the thing to work at all has been agony. The idea that there are faults in it does not bear thinking about.

And if you actually uncover faults, it means that you will have a lot more work to do and unpick lines of code that, as much by luck as anything, seem to work at present. Like it or not, the only way to test your programs and have a good chance of uncovering bugs is to prepare the test data beforehand.

Preparing test data

A program is to be written which will read a list of examination marks typed in at the keyboard and find the average examination mark. The program should reject examination marks which are not within a specified range and provide a suitable error message to the user.

1. Write down what you think an acceptable range for an examination mark would be;
2. Write down an algorithm, in pseudo-code notation, to represent a solution to this problem;
3. Copy the headings in the table below and construct at least 5 test data entries to demonstrate how you would verify the correctness of the program.

Test Case	Reason	Expected Result	Actual Result	Comments

Sentence completion - test data

On the Web is a interactivity. You should now complete this task.

3.4.2 Review questions

Q11: Which one of the following options states the main purpose of test data?

a) To determine that the system meets the specification
b) To prove the absence of errors
c) To show that the programmers have been careless
d) To minimise the number of errors in the program

Q12: Error detection can be time-consuming. Which one of the following could be added to a program to help detect errors?

a) Suitable commentary throughout the program
b) Output statements at key points in the code
c) Specific error-detecting code
d) Nothing can be added

Q13: Choose the statement that best describes testing for errors:

a) There is a certainty that undetected errors will always exist
b) Testing can never show that a program is correct
c) Testing can demonstrate the absence of errors
d) All of the above

Q14: One example of a fault-avoidance technique in developing software is?

a) Take more time to design better software
b) Hire more programmers so that errors will be easier to detect
c) Design input routines that will not crash when presented with unexpected data
d) Make sure that that the code is specifically written to avoid errors

Q15: The last remaining errors in a program are not easy to remove because

a) They could remain hidden until the program is run under all conditions
b) The compiler is not very efficient
c) De-bugging procedures are not effective enough
d) They will not affect the running of the program since most errors have been found

3.5 Structured Listing

A structured listing is a hard copy of the program source code. It is important that the source code is laid out in accordance with the conventions of the implementation language. The code should be properly indented; this helps people follow the structure of the code. Meaningful names should be used for modules, constants, and variables. The use of **internal commentary** will help to explain the logic of the code to others and also serve as a documentation aid for the programmer. A structured listing can be produced at any time during implementation. It can serve as a tool for checking program logic and also form part of the final software documentation.

Figure 3.6 shows a structured listing of a program coded in the language Visual Basic.

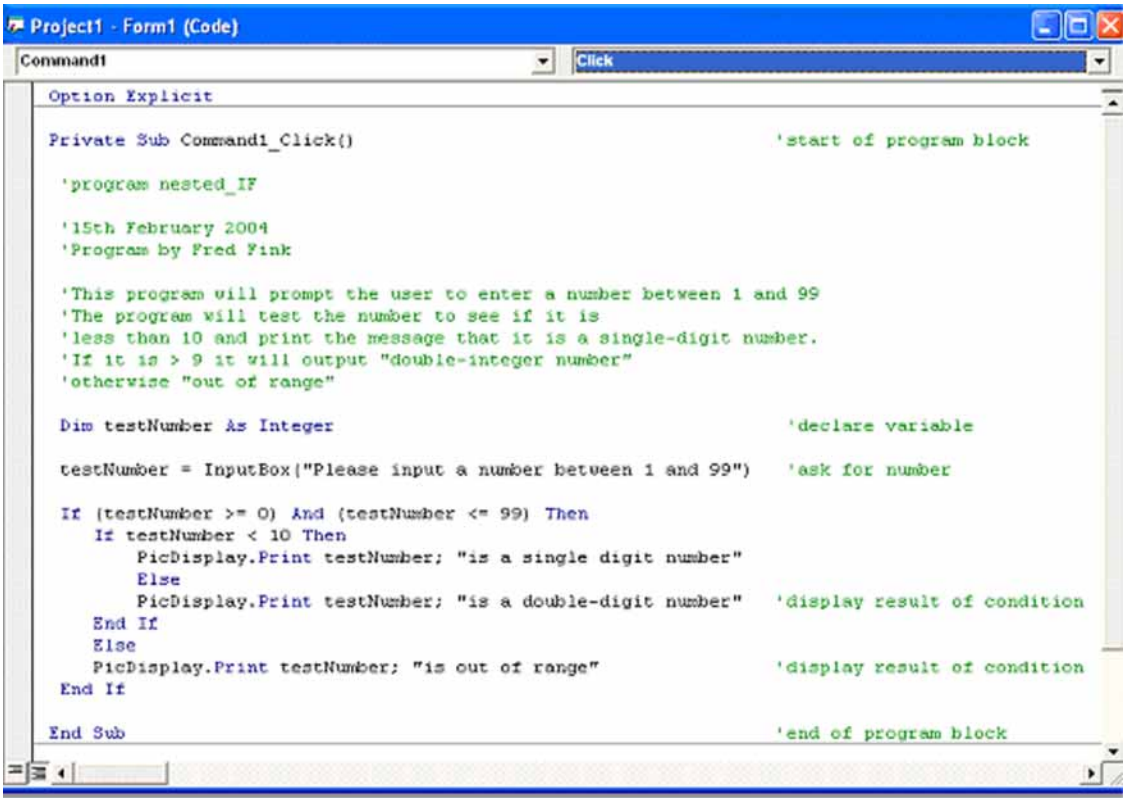


Figure 3.6:

3.6 Error Reporting

Errors can occur at any of the stages in the development process.

Errors can be categorised in many ways. This section will describe the following 3 categories of error:

- compilation
- execution
- design

3.6.1 Compilation Errors

This is an error which is detected during compilation; *eg a syntax error* . Examples of typical errors that can occur at the compilation stage include:

- incorrect use of programming language syntax
- wrong *number* of parameters used in the call to a module
- incorrect *type* of parameter passed to a module
- reference to undeclared variables.

3.6.2 Syntax Errors

These are errors which result from incorrect use of the programming language structure; *ie incorrect use of the language grammar*. They are detected during compilation and examples include:

- use of programming language keywords as variable identifiers
- blocks that have a missing "end" marker - e.g in Visual Basic a missing `End If` statement
- missing brackets/semi-colons.

These types of error can be very irritating, as they prevent the generation of object code. The program cannot run. Those new to programming often take a long time to become familiar with the syntax of a new language, to the extent that they can write code as fluently as is it were the English language.

3.6.3 Execution / Run-time Errors

These are errors detected during the execution of the program; *eg division by zero*. They are also known as run-time errors. Even given that a program is translated into machine code and runs on the computer, there is no guarantee that it will not generate errors. For example, a common mistake made by programmers is to divide by zero. In most cases this will produce a run-time error and cause the program to halt unexpectedly.

Another example includes attempting to read character data directly as numeric.

'Array Bounds Exceeded' is a common mistake in languages which support array data structures. Here, the programmer is attempting to access a position within the array which exceeds its predefined bounds.

3.6.4 Logical Errors

These are errors in the design of the program; *eg calling the wrong procedure or routine.*

Even given that the program translates to machine code and does not halt unexpectedly due to run-time errors, there is no guarantee that it will not fail. It may contain logical errors. These are errors in the logic of the code itself. For example, writing code to add two numbers instead of multiplying them, or forgetting to write code to do something when a condition that is being tested fails.

Other examples include:

- making a call to the wrong module
- passing incorrect data into a module
- passing the wrong data out of a module.

3.6.5 Coding Errors

At implementation there are a number of error types that can occur. You have already seen how translators will fail to convert the source code to object code if the grammatical rules of the programming language is not adhered to.

Other errors typical at the coding stage include:

- initialisation errors - variables are not assigned an initial value
- loop counter errors - doing something too many times or not enough/failing to increment a loop counter
- failure to define or declare variables
- incorrectly declaring structured data types and/or dimensions (in the case of arrays).

3.6.6 Design errors

Errors do occur at the design stage and some of the more typical ones include:

- incorrect interpretation of program specification
- incomplete logic
- neglect of special cases
- poor error handling.

As you have already learned, errors at the design stage can be very costly to correct. Many months of development may have already been undertaken. The re-design of software can introduce more errors as existing code is altered to meet the specification. As we have stated at the beginning of this section, fixing errors can be very expensive because as faults are removed the cost of finding and removing remaining faults increases exponentially

3.6.7 Dry running

One goal when preparing software is to minimise the number of errors introduced into the system. A number of techniques can be used to help. These include:

Use of static testing techniques (also called **desk checking** and dry-running). These do not require the program to be run but instead examine the source code of the problem. This has the advantage that each error can be considered in isolation, and the error interactions are insignificant. This is a surprisingly effective technique.

When using these techniques, we strongly suggest that you examine a printout of the program, and do not fiddle with it on screen. If you are looking at a screen version the temptation is to make changes without thinking enough about their consequences. So you go from a program which does not work well to a program which does not work at all. At this point you find that you have probably not got a back-up copy, either.

Additional approaches include working with others to help identify errors. For example, reading code out to others in a group, or explaining code to other people

The problem is that many bugs are so hard to fix. Staring at the screen and tinkering with the code does not help much with the more complex problems. One reason for this is that modules are defined in one part of the code and called in another: to check the progress of the program's execution, you have to keep paging up and down. It is often much simpler to get a listing of the code and settle down to work with that. You sort out the listing so that the relevant modules and module calls are in front of you, and you do a static test of the program. You ask yourself, "What should it do next?", *"What state should the data be in?"*

These are the most common errors to look for:

- are all variables initialised before use? - remember that some translators do not initialise variables to zero
- are all constants named?
- do all loops terminate?
- is the end of an array overwritten? (a very common mistake in programming)
- are the conditions correct in conditional statements?
- are there misplaced, or missing, statement terminators?

3.6.8 Trace facilities

Trace facilities are built into many languages. A trace facility is used to identify the path taken through the program. Trace facilities may help to identify logical errors.

The facility is implemented by toggling between TRACE ON and TRACE OFF or by running a debugging facility which will allow the program to be run step-wise, line by line.

3.6.9 Review questions

Q16: A compilation error is best described as (choose one)?

a) An error which occurs during execution of a program
b) An error generated by the source translator
c) An error in the logic of the code
d) An error made by the translator

Q17: When carrying out a dry run which one of the following would represent common errors to look for?

a) Do all program loops terminate?

b) Has the programmer used meaningful variable names?
c) Has the programmer used conditional statements?
d) None of the above

Q18: From the following list identify one other error to account for during a dry run:

a) Initialisation of variables
b) Naming of constants
c) Correct conditional statements
d) All of the above

Q19: A trace facility in a program will identify (choose one):

a) Compilation error
b) Syntax error
c) Logic error
d) Coding error

Q20: Program errors can be categorised into types. Which one of the following is NOT an error type?

a) Compilation
b) De-bugging
c) Execution
d) Design

3.7 Module libraries

The use of previously written modules reduces the time spent creating the final software and minimises the cost. Designers will incorporate, if possible, modules from the module library in the design.

For many software companies and programming departments, one project will be similar to another. Projects will have component parts which are similar; for example many programs might require a sort routine to act on the contents of a database.

For many software companies and programming departments, one project will be similar to another. Projects will have components parts which are similar; for example many programs will require a sort routine to act on the contents of a database.

As time goes by, a collection of modules is built up. This collection is known as a **module library**.

Program libraries or software libraries are made available for common use. It may contain compilers, utility programs or code fragments to perform specific operations.

Modules in the library can often be used, over and over, in new projects. This saves time spent designing and coding. If a module can be used without alteration, there is the added advantage that it has been thoroughly tested and is reliable. Use of unaltered modules reduces the time spent on the detection and correction of errors in the project as a whole.

Sometimes adjustments might be needed to a library module. After the necessary work,

the module will require to undergo error and other testing.

Characteristics of module libraries

On the Web is a interactivity. You should now complete this task.

3.8 CASE tools

Programmers also use Computer-Assisted Software Engineering (CASE) tools to automate much of the coding process. These tools enable a programmer to concentrate on writing the unique parts of the program. CASE tools generate whole sections of code automatically, rather than line by line. This produces more reliable and consistent programs and increases programmers' productivity by eliminating some routine steps.

When one is designing (or re-designing) a software system, it's important to be consistent and not to break any of the rules. For example:

- inputs and outputs must be preserved from level to level:
- names must not be duplicated.

If the system is at all large, the difficulties involved become significant.

CASE tools are used to help look after the (very important) details.

For example, when we're refining a process, the CASE tool will deal with the input and output flows automatically. Likewise, it may provide extensive support for drawing diagrams.

The CASE tool will also prevent the use of the same name for different things in the course of building up a data dictionary for the system.

3.9 Software characteristics

This section introduces the terms robustness, reliability and portability.

A program is robust if it can cope with problems that come from outside and are not of its own making e.g. *corrupt input data*. Reliability is an internal matter. A program is reliable if it runs well, and is never brought to a halt by a design flaw.

When the program is complicated the distinction between the two terms is not always clear. When a machine hangs it is not always obvious whether this is due to a failure in robustness or reliability.

3.9.1 Robustness

The designer should try to ensure that the design is **robust**: the resulting software should be able to cope with mistakes that users might make or unexpected conditions that might occur. These should not lead to wrong results or cause the program to hang.

As examples of an unexpected condition, we could take something going wrong with a printer (it jams, or it runs out of paper) or a disc drive not being available for writing, because it simply isn't there (the user's forgotten to put in the floppy disc).

3.9.2 Reliability

A **reliable** program is able to react to the unexpected without crashing.

It should possess the ability to cope with errors and unexpected events during its execution. Possible outcomes of error encounter could be:

- inform the user
- die gracefully
- recover reasonably within a minimum time period

3.9.3 Portability

A portable program can run on a variety on machine architectures under different operating systems with little or no modification. A portable program is one which is machine independent.

However the problem of porting programs from one operating system to another still exists. The problem became acute with the development of the Internet, which is meant to enable communication between all sorts of computers and operating systems.

A solution was devised by Sun Microsystems which introduced another layer of software, called **Java**. Programs written in Java are compiled into bytecode that can be run using a Java interpreter. This is available for different machine platforms eg Apple Macs, PCs etc.

A machine with a Java interpreter installed is Java-enabled. A Java-enabled machine can run Java bytecode produced on any another kind of machine. For example, you could create the bytecode on a PC running Linux and this will run under Windows or on a Macintosh computer.

This approach works very well for small programs or applets on the Web, but interpreters tend to run slowly, so the porting of large computer programs remains a problem.

A large, and successful, commercial system will generally outlast the hardware it was developed on. When the hardware is changed, the system will need to be transferred, or ported, to the new hardware.

If a program works well in one part of an organisation, the decision is often made to set it up in other parts, where, for one reason or another, the hardware might be different.

3.9.4 Review questions

Q21: Choose the correct response that relates to the term **robust** within software development:

a) The program is strong and hardy
b) The program may be ported to a different machine architecture
c) The program can cope with mistakes that the user might make

d) The program runs to specification

Q22: Using a module library can reduce software development time because (choose one):

a) Common programming modules can be used
b) Programmers do not need to write programs from scratch
c) Programmers can re-use library code in their projects
d) All of the above

Q23: An example of an unexpected condition that could cause a program to hang until user intervention would be (choose one):

a) A corrupted floppy disc
b) A printer jam
c) A damaged keyboard key
d) A pointing device not attached

Q24: Which one of the following actions will a reliable program NOT do?

a) Inform the user that something is wrong
b) Produce output, come what may
c) Recover from an error situation within a minimum time period
d) Die gracefully

Q25: A language designed to be portable across different operating systems is (choose one)?

a) Visual Basic
b) Pascal
c) Java
d) C++

3.10 Summary

The following summary points are related to the learning objectives in the topic introduction:

- much has to be done in terms of design and testing of software before the coding and implementation stages;
- various graphical design constructs are available;
- top-down design and stepwise refinement is a well-established technique;
- testing is an extremely time-consuming process;
- the main aim is to produce software that contains the basic elements of reliability, portability, efficiency and maintainability;
- laudable as the efforts in the various phases of the software development process might be, no piece of software can be considered to be error-free.

3.11 End of topic test

An online assessment is provided to help you review this topic.

Topic 4

Personnel

Contents

Prerequisite knowledge

There are no prerequisites for this topic.

Learning Objectives

- *Identify the personnel at each stage of the Software Development Process*
- *understand the role of each person*

4.1 Introduction

The Software Development Process involves many people throughout a computer system's lifecycle. In this topic you will meet the personnel involved, with a brief outline of their activities.

4.2 Personnel

The software development process is initiated by one of two processes:

- either an old system is not working well or
- an altogether new system has to be set up.

A new system is created by a project. The creation of this, of course, is the work of people, and it is important to say something about the personnel involved in the process.

Although many people may be involved, the key personnel are discussed under the five main headings:

1. client
2. project manager
3. systems analyst
4. programmers
5. independent testing group

In a nutshell the process encompasses the following events:

The management who require the new or updated system represent the clients. They approach external consultants, the people who will create the system. The consultants appoint a **project manager**, who carries out a feasibility study. If the feasibility study bodes well, the management asks for a full system investigation. This is carried out by a systems analyst from the consultants, who works with the project manager. It culminates in an operational requirements specification.

When the operational requirements have been agreed by both the consultants and the customer, a contract for the system is drawn up. The consultants put a team of programmers on the job to code the specification, and the software development process begins. Once the software has been implemented in a chosen language it undergoes rigorous testing by an **independent testing group**.

4.3 The Client

In this case the client is represented by the management of the company Management are responsible for overall business and strategic planning. They choose the projects

that will go ahead. They review the progress of a project.

Obviously a project should benefit the organisation in some way, but it's not always easy to assess and compare the benefits that different projects might bring.

One project might be relied upon to break even or show a small profit, another might not be reliable at all but might make an enormous profit. A project might not make much in itself but be worth doing because it could lead to future projects that would be immensely profitable. Many projects are carried out not directly to make profit but to improve things in the organisation. Non-profit making organisations often have to assess projects in terms of the benefits they will confer rather than the money they will bring in.

It is important to note that the terms client and user do not necessarily mean the same thing, but are sometimes used interchangeably. A client is someone or a group such as management who buys or intends to buy some software for a particular purpose. A user is someone who uses or will use the software.

Acting as the client or customer for the organisation the appropriate personnel will be concerned with the following activities:

- holding discussions with the consultants
- review reports, and negotiate contracts
- ensure the provision of relevant information to the project manager and other members of the consultancy team
- deciding whether or not to carry on
- paying the consultants

The client, of course, can't simply leave things up to the consultants. The client must provide detailed direction to develop the project. They have to promote support for the study within the organisation and ensure that the relevant resources are available to the investigators. They must make sure that people with a stake in any possible development are involved in the study. They must agree with the investigators on the description of the problem and on the assignment of priorities. They represent the organisation as client, and must make sure that all relevant information is given to the consultants. When the study is completed, they have to evaluate the possibilities presented.

Sentence completion - software development

On the Web is a interactivity. You should now complete this task.

4.4 The Project Manager

The project leader is responsible for the project. So far as these notes are concerned, the project manager will be appointed by the consultants.

The project manager supervises the project and carries out the initial stages. If it goes ahead, the project manager will take charge of the project, from the first brainstorming

session to the software launch and implementation.

It's up to the project manager to keep the process on schedule by whatever means possible. Sometimes it may involve 'cracking the whip' to make sure the work is up to standard and that important deadlines are met - being a project manager means being accountable for the entire duration of the project.

Project managers are easy targets for criticism because the success or failure of a project is often dependent on their decisions. Hence it can be a thankless task at times for this overworked person!

The most time-consuming undertaking of all will be managing people whether they be clients or as members of the consultancy team.

With clients, the project manager:

- needs to extract all the on-going relevant information from the client and force him/her to make decisions regarding software specification
- has the ability to accommodate substantial changes to the original specification made by the client from the original specification and offer revised schedules and budgets at short notice.

With his/her own team, the project manager:

- adopts a counselling role for the members, offering therapy to disgruntled or stressed-out employees
- promotes good welfare by raising team spirit through regular, personal meetings and positive appraisal.

A job description for a typical project manager might involve the following requirements:-

- have experience in dealing with a variety of clients
- the ability to manage a team
- possess excellent verbal and written communication skills
- have strong attention to detail
- possess the ability to create schedules and budgets
- show remarkable endurance under stress

4.5 The Systems Analyst

The systems analyst carries out the system investigation. (In large systems, the analysis might be carried out by a team of analysts.)

The systems analyst is appointed by the project manager.

Systems analysts are active in the design, testing and implementation phases of the project. They often work in a team, with significant liaison with external or internal clients.

Systems analysts will usually have learnt some programming but they won't necessarily be programmers. It could not be taken for granted that an analyst would know anything about the programming language in which the system will be implemented

Most systems analysts work with a specific type of system that varies with the type of organization they work for. Examples would include business, accounting, or financial systems, or scientific and engineering systems. Some systems analysts also are referred to as **systems developers** or systems architects.

A requirements specification is the starting point and the analyst must proceed from there. The aim is to produce a clear specification that the rest of the development team will use in the subsequent stages.

Analysts begin an assignment by discussing the systems requirements with company managers (clients) and users to determine the exact nature of the problem. They define the goals of the system and divide the solutions into individual steps and separate procedures.

Full systems analysis has three phases:

1. collection of information
2. analysis of information collected
3. production of a problem specification or user requirements specification.

Identifying Personnel involved in a Systems Development

On the Web is a interactivity. You should now complete this task.

4.5.1 Collection of Information

There are various methods used for gathering information. These include:

(a) Interviews.

The analyst talks face to face with clients, to find out how the current system works and what is required of the new system.

(b) Questionnaires.

Where the current system has a large number of users, the analyst might construct a questionnaire for everyone involved. People often respond more frankly to an anonymous questionnaire. On the other hand, the response rate can be low.

(c) Observation.

The analyst studies the current system and observes how it works. This is useful for bringing to light things that users take for granted.

(d) Document analysis.

Many different kinds of document are involved in a system: the documents that the system produces; the documents it uses i.e source documents and the documents that

affect how the system works (such as documents that spell out the procedures to be followed in using the system). The first gives the analyst an idea of what the new system will have to produce, and the second will help understanding of the workings of the current system.

(e) The What-Where-When-Why-Who approach.

The analyst seeks answers to the obvious questions beginning with these words: *what* does the system do, *where* are these things done, *when* are they done, *why* are they done, and *who* does them?

Many of these techniques are essentially iterative. The answers to questions often raise further questions, which the analyst must go and ask, and so on.

(f) Expert Knowledge

The analyst has to try to gain understanding of the processes the new system is supposed to help.

This understanding is important because particular people tend to take their own special knowledge for granted and as a result don't mention important requirements to the analyst.

A basic understanding of the process will allow the analyst to ask more searching questions during the interviews with key informants.

4.5.2 Analysis of the information

When the analyst considers that enough information has been gathered, work starts on the analysis. However, it may well be that the analysis reveals shortcomings in the information so far obtained. In such a case, iteration arises again: the analyst has to go back for more information.

Having gathered information, the analyst has to understand it. All computer systems have three sub-systems:

- input
- processing
- output

The basic question is how to process the input to get the required output.

In small, straightforward cases, this can be answered by studying the information and giving it some thought.

More complicated cases are easier to understand if the analyst creates a model of the system. This model focuses on the essential workings of the system and the connections between the component parts. Common approaches to system modelling are data-flow modelling, object modelling, structured analysis and information engineering. These methods specify the inputs to be accessed by the system, design the processing steps, and format the output to meet the users' needs. They also may prepare cost-benefit and return-on-investment analyses to help management decide whether implementing the proposed system will be financially feasible.

Many designers consider that priority should be given to the data, *eg in cases where programs are needed to maintain a database of records or stock keeping*. Information engineering is a data-centred technique that focuses on the data and uses structured design techniques to create the processes.

4.5.3 Review Questions

Q1: In the development process the client representative is best described as (choose one):

a) The group who will use the software
b) The group who will test the software
c) The group who will purchase the software
d) The group who will write the software

Q2: In a company the idea of starting a new project is mainly to:

a) Make as large a profit as possible
b) Use up surplus capital otherwise it will diverted elsewhere
c) Give the users in the company something else to do
d) Benefit the organisation in some way

Q3: Which one of the following statements does not refer to the systems analyst?

a) The systems analyst is appointed by the project manager
b) The systems analyst discusses the system requirements with the clients
c) The systems analyst determines the exact nature of the problem
d) The systems analyst is responsible for the entire project

Q4: There are three phases associated with systems analysis. One of the main reasons for this is to:

a) Allow the systems analyst to produce a clear specification of the problem
b) Allow the client to say how much the project will cost
c) Allow the project manager to dictate whether the project should go ahead
d) Allow the collection of information that the company will need anyway

Q5: Should a project fall behind schedule then it is up to one of the following to get it back on track:

a) The client
b) The project manager
c) The systems analyst
d) The project will eventually recover itself

4.5.4 Production of the problem specification

When a system is accepted, the system analyst will determine what computer hardware and software will be needed to set it up. He/she will coordinate tests and observe initial use of the system to ensure it performs as planned. They prepare specifications, work diagrams, and structure charts for computer programmers to follow and then work with them to "debug," or eliminate errors from, the system. Analysts, who do more in-depth testing of products, may be referred to as *software quality assurance analysts*. In

addition to running tests, these individuals diagnose problems, recommend solutions, and determine if program requirements have been met.

Once the analyst has arrived at a clear idea of the problem, this is expressed in a document called a specification. It may be called the problem specification, requirements specification or user requirements specification.

The specification includes a full description of the problem. All the inputs, processes, and outputs are described. No system works on its own, independently of the outside world: certain assumptions have to be made about the boundaries between a system and its environment, and these must be indicated, described and included.

In some cases, the specification represents an agreement between the clients and the development team. The process of reaching agreement will be iterative: the analyst will present a draft specification to the clients, who will suggest amendments, and so on, until a specification is agreed.

The specification is used throughout the rest of the development process. Material produced is based on it and compared with it. For example, test data will be drawn up on the basis of the specification.

The specification can be used as a checklist, to ensure that the development process remains on target.

A systems analyst's main task can be summarised as follows:

- translating client requirements into highly specified project briefs
- identifying options for potential solutions and assessing them for both technical and financial suitability
- presenting proposals to clients
- working closely with programmers and a variety of end users to ensure technical compatibility and user satisfaction
- ensuring that budgets are adhered to and deadlines met
- drawing up a testing schedule for the complete system
- overseeing the implementation of the new system
- providing training to users of the new system

4.6 The Programming Team

The programming team is responsible for the second part of the project creation, the coding phase of the software development process. They work to the design created by the systems analyst. They are also responsible for testing the software, and for maintaining it once it has been installed. The team will report to the systems analyst.

As a rule, at least in large projects, the programming team reports to the systems analyst and tends not to have much contact with the project manager.

The analyst schedules the work, keeps an eye on performance, and oversees the

development of the system.

In respect of production, a programmer's work is in two parts:

First the detailed logic of the modules in the system has to be worked out. Data flow has to be identified at all stages.

Second, the programmers write the code, test it and debug it-though testing is often done by a separate team.

The programmer is involved at all stages of integration of modules, and follows a designed test strategy throughout.

In most cases, several programmers work together as a team under a senior programmer's supervision.

Programmers often are grouped into two broad types-applications programmers and systems programmers. Applications programmers write programs to handle a specific job, such as a program to track inventory, within an organisation. They may also revise existing packaged software. Systems programmers, on the other hand, write programs to maintain and control computer systems software, such as operating systems, networked systems, and database systems. These workers make changes in the sets of instructions that determine how the network, workstations, and central processing unit of the system handle the various jobs they have been given and how they communicate with peripheral equipment, such as terminals, printers, and disk drives. Because of their knowledge of the entire computer system, systems programmers often help applications programmers determine the source of problems that may occur with their programs.

Programmers will update, repair and modify existing programs. When making changes to a section of code, they make other users aware of the changes by inserting comments in the coded instructions so others can understand the program.

Programmers test a program by running it, to ensure the instructions are correct and it produces the desired information. If errors do occur, the programmer must make the appropriate change and recheck the program until it produces the correct results. This process is called debugging. Programmers may continue to fix these problems throughout the life of a program.

4.7 Independent Testing Group

Consider the following quote from computing expert in the field of software development:

"No, no, no, no, no.....the people who develop and implement code should have absolutely no role in testing that code! A test team should be TOTALLY independent from the implementation team..."

Quite an adamant view with a clear message!

Programmers would be less inclined to test their software to destruction and take into effect test data that might cause unexpected results. - they have the knowledge of how the functionality of the code was designed and will test accordingly. Also they probably don't have sufficient time to fully test the program under all conditions.

The software is, therefore, passed to other groups of people - independent test groups who will undertake impartial testing.

Alpha testing

This is where the software, usually with minimum functionality at this stage, is passed to personnel in-house who will undertake full, exhaustive testing under varying circumstances. The test group focus on the implementation of the program specification, and provide feedback to the programmers on specific aspects of the software.

Beta testing

From the alpha testing phase, beta testing represents the pre-release version of the software which is made available to a large number of selected users in the outside world. The software will be run on a variety of computer platforms under real conditions.

After successful beta testing the software is then released for general use, usually with a version number.

For more on beta testing have a look at the following web site:

http://beta.intuit.com/public/upcomingBetaInfo.cfm

Roles within Software Development

On the Web is a interactivity. You should now complete this task.

4.7.1 Review Questions

Q6: The program specification is an important document in the software development process. The main reasons for this is (choose one):

a) It can be used as a checklist to ensure the project is on target
b) Test data will be drawn up on the basis of this document
c) It describes all the inputs, outputs and processes involved in the project
d) All of the above

Q7: Once the program has been designed it is then passed to the programming team. Which one of the following actions is not performed at this stage?

a) They report directly to the project manager at all stages of programming
b) They are able to modify and repair existing programs
c) They are responsible for all test strategies concerning the software
d) They are overseen by the systems analyst at all stages of their work

Q8: The completed project is usually tested by an independent group. This is because (choose one):

a) Independent test group will have better facilities for testing software
b) Programmers will tend to test only within the functionality of their own code
c) Independent test group will not test further than they have to so saving time
d) Programmers are happy to test their programs to destruction

Q9: If a project begins to run over budget then this will be the responsibility of one of the following personnel to change this state of affairs:

a) The client
b) The programming team
c) The systems analyst
d) The project manager

Q10: In beta testing, which one of the following is true?

a) Testing is done in-house
b) Testing is done by specialist personnel at cost
c) Testing is done by external groups on a variety of computer platforms
d) Testing focuses on the problem specification

4.8 Summary

The following summary points are related to the learning objectives in the topic introduction:

- personnel involved in the software development process;
- various levels professional expertise required at each stage of the process;
- it is an iterative process involving on-going communication at all stages;
- the final program may not meet client specification for various reasons.

4.9 End of topic test

An online assessment is provided to help you review this topic.

Topic 5

Languages and Environments

Contents

Prerequisite knowledge

Before studying this topic you should be able to:

- *describe and compare machine code and high level languages;*
- *explain the need for translation of high level language;*
- *describe the function of a compiler;*
- *describe the function of an interpreter;*

- *describe the process of recording a macro;*
- *assign a keystroke to a macro;*
- *describe examples of the use of macros;*
- *describe the features of a text editor.*

Learning Objectives

- *understand the differences between procedural, declarative and event-driven languages*
- *be able to describe the uses of compilers and interpreters*
- *understand the functions and efficiencies of compilers and interpreters*
- *be able to describe the features and uses of scripting languages*
- *understand how to create and edit a macro*
- *understand the need and benefits of scripting languages*
- *be able to describe the use of module libraries*

Revision

Q1: Programs are written in high level languages and then translated before they can be executed by a computer. This is because:

a) Computers do not understand English words
b) It is easier to write high level code
c) Computers only understand machine code
d) All of the above

Q2: Two translator programs are a compiler and an interpreter. Which one of the following statements is true?

a) A compiler produces object code whereas an interpreter is much faster
b) A compiler is much faster whereas an interpreter executes a line at a time
c) An interpreter produces object code whereas a compiler is much faster
d) An interpreter is much slower whereas a compiler executes a line at a time

Q3: Which one of the following error situations would be picked up by a compiler or interpreter?

a) Run time error
b) Logic error
c) System error
d) Syntax error

Q4: A problem may be implemented by using either a procedural language or a declarative language. Which one of the following statements is true?

a) Pascal is a procedural language used in education
b) Comal is a declarative language used in artificial intelligence
c) Prolog is a procedural language used in artificial intelligence
d) Visual Basic is declarative language used in education

Q5: A scripting language differs from a macro in that:

a) A scripting language is difficult to use
b) A macro must be run by pressing a key
c) A scripting language has to be translated
d) A macro can be used to automate simple tasks

5.1 Introduction

This topic will introduce you to the various types of programming languages that would be available at the implementation phase of the software development process. As you will see the choices are numerous and expertise is required in matching the system specification with the most appropriate programming language available. How programming languages translate source code into object code is further explained, emphasising the advantages and disadvantages of the methods used.

5.2 Programming Languages

In this topic and others, all language features are exemplified using Visual Basic

During the implementation phase of the software development process the program specification will now be coded using a suitable high level programming language. High-level languages help software developers to identify more with the problem rather than the hardware on which the final program will run i.e. they are **problem oriented**. Program statements and expressions in such languages generally incorporate English words which mean that they are easier to understand and use than assembly code.

There are many programming languages that could be used for the implementation phase of software development, such as C and C++, Pascal, and Java. Older languages still much used include Cobol and Fortran. Cobol is used mostly on main frames for batch processing: organisations still use it for new programs because all their old programs are written in it. Fortran is still used for applications where the emphasis is on numerical processing (such as processing meteorological data.)

Programming languages, however don't remain unchanged. They develop due to a variety of reasons; people perceive shortcomings in a language and try to correct them; the world moves on and people try to adapt a language to cope with new demands and expectations. Some changes are formalised with new versions of a language being agreed and released from time to time.

The choice of language may be based on:

- the experience and expertise of the development team
- the range of languages and development platforms available to the organisation
- which language has the facilities most appropriate to solving the required problem
- the choice of a particular language substantially reducing the overall development time
- a suitable compiler/interpreter available for the client hardware.

If the task requires text processing, for example, a language which supports data of type string is necessary. In other circumstances a language which supports arithmetic, logical operators, sound or graphics may be necessary.

Another factor is portability. A program is portable to the extent that it can be used on different computer hardware. If the programs are in the form of machine code a program compiled into machine code on a PC will not be executable on a different machine, eg *Apple Macintosh*.

5.3 Classification of High Level Languages

At the time of writing it has been estimated that around 2,400 programming languages have been catalogued over the years.

For further information and a chance to download a time line of language development

as a poster, consult the following links:

http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm

http://www.levenez.com/lang/history.html#08

Classification of programming languages is fraught with problems since, as you will see, many languages can fall into more than one category.

Historically programming languages were classified according to whether they were a **general purpose language** that could be applied to a broad range of situations, or a **special purpose language** that were designed for specific tasks.

Table 5.1 summarises the main high level languages and variants:

Table 5.1: Historical classification of programming languages

Area	**Language(s)**	**Purpose**
General purpose	ALGOL , ALGOL 60, ALGOL 68 *(ALGOrthimic Language)*	Coding of general algorithms. ALGOL 60 was the first defined language i.e. code produced identical results on any mainframe computer world-wide. Also introduced procedural programming and parameter passing. Derivations are: Pascal, C, C++ and COMAL
Scientific	FORTRAN *(FORmula TRANslation)*	Developed in the 1950's for use in scientific and engineering applications and is still in use today.
Commercial	COBOL*(COmmon Business Oriented Language)*	Suitable for data processing applications. So widely used, COBOL 97 is the latest version in use.
Education	Pascal	Developed from ALGOL, developed in the 1970's to teach structured programming
	BASIC (Beginners All Symbolic Instruction Code)	One of the first interpreted languages, designed for beginners to programming in the 1960's.
	COMAL (COMmon Algorithmic Language)	The language promoted the use of structured code as opposed to the 'spaghetti code' from using BASIC.
Artificial Intelligence	PROLOG*(PROgramming in LOGic)*	Used in the construction of AI applications, expert systems and in the teaching of AI.
Operating systems	C	Derived from UNIX as the 'C' shell it now has many uses in programming, being very close to assembly language.

Alternatively high level languages can be classified according to their structure and purpose. Although quite an extensive list, for the purposes of this topic the following categories are of importance:

1. Imperative/procedural
2. Declarative/logical
3. Event-driven
4. Scripting

   Two other types that tend to be associated with the above categories in present-day programming techniques are:

5. Object-oriented
6. Functional

A table based on this language classification is shown Table 5.2

Table 5.2: Classification based on structure

Programming Language	**Structure**	**Purpose**
Pascal	imperative	general purpose language, widely used
Visual BASIC	imperative	windows interface applications, multimedia
PROLOG	declarative	artificial intelligence
Visual C++	event-driven	used as front end to develop user interface
COBOL	imperative	business use
Java	object-oriented	platform independent - an object oriented language
VBscript	scripting	creating and editing macros
FORTRAN 90	object-oriented	scientific and software engineering
Lisp	functional	artificial intelligence uses - many other languages are derived from it
JavaScript	scripting	writing and enhancing web pages
BASIC	imperative	easy to learn, originally developed to teach non-specialists the art of programming
FORTRAN	imperative	scientific programming language

The above list is not in any way rigorous. There are many instances of particular languages fitting into more than one category. For example, although Visual basic is listed as an imperative language it can also be classed as an event-driven language and also an object-oriented language.

5.4 Imperative languages

Imperative languages are also known as **procedural languages** because they employ structures which include procedures and functions. Programs generated in procedural languages involve a sequence of operations and are often described as linear programs.

A procedural (imperative) programming language tells the computer *how to do something*, written as an ordered sequence of steps that describe exactly what it must do at each step. These instructions, which form the basis of an **algorithm**, are followed in written order by the computer.

Three basic constructs are used to define the order of the steps:

1. *sequence* (the logical ordering of steps);
2. *selection* (a step or sequence of steps are performed if a condition or set of conditions is true);
3. *iteration* (a step or sequence of steps are carried out repeatedly).

Both iteration and selection are *control* constructs because they can alter the *flow of control* of program execution. You will see more of this in later topics.

Examples of imperative languages are Algol, Fortran, Pascal, Basic, C and COBOL.

Example of a program to read data into an array and output results

Problem: In Visual Basic we might tell the computer how to input data into an array and then print out the results:

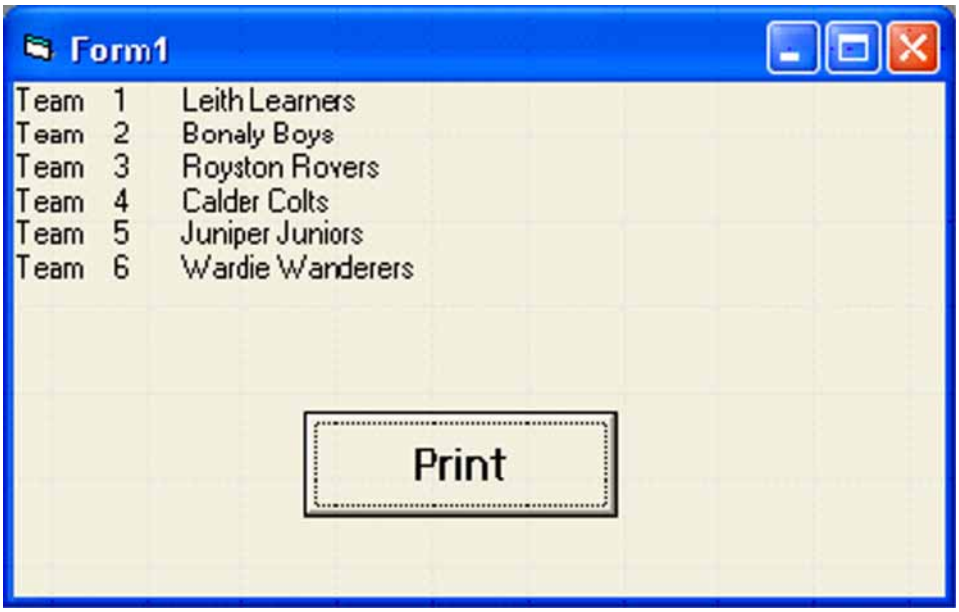
Solution:

```
'Program to fill an array with team names and print them out

Private Sub Command1_Click()
Dim Team(5) As String
Dim Count As Integer

'Assign array values

Team(0) = "Leith Learners"
Team(1) = "Bonaly Boys"
Team(2) = "Royston Rovers"
Team(3) = "Calder Colts"
Team(4) = "Juniper Juniors"
Team(5) = "Wardie Wanderers"

For Count = 0 To 5
  Print "Teams"; Tab(8); Count+1; Tab(14); Team(Count)
Next Count
Print

End Sub
```

Notice that:

1. the program is expressing at each step precisely how each statement is executed;
2. the program has beginning and end points, which is an indicative feature of imperative coding.

The program output is:



Structures in procedural languages

On the Web is a interactivity. You should now complete this task.

5.4.1 Review Questions

Q6: One of the main reasons for the many programming languages existing is:

a) Older languages can no longer function in the present day computer systems
b) Languages have to be adapted so new versions are released
c) There are many different types of operating system
d) People devise new languages because they do not like existing ones

Q7: Which one of the following lists only contains general purpose programming languages?

a) Pascal, C, Basic,Comal
b) Algol, Fortran, Prolog, Comal
c) Basic, Algol, Pascal, Comal
d) Fortran, Basic, Comal, Algol

Q8: In computer programming there are three basic constructs. Which one of the following is NOT a programming construct?

a) Sequence
b) Selection
c) Iteration
d) Moderation

Q9: Which one of the following languages could be classified as being close to assembly language?

a) Cobol
b) C
c) Visual Basic
d) Prolog

Q10: Choose one of the following that could refer to imperative languages:

a) They employ procedures and functions
b) Programs written in the language are linear in structure
c) Programming instructions are explicit
d) All of these

5.5 Declarative languages

Declarative languages or Logical languages model problem solutions very differently. Programmers specify what the problem is rather than how to solve it. In PROLOG, for example, a program represents propositional knowledge as facts and uses rules to test facts, building more complex propositions. Collectively facts and rules are called clauses. A proposition is the smallest unit of knowledge that can be judged true or false, such as "*a collie is a sheepdog*", or "*a beagle is a hound*" or "*George passed a ball to Steven*".

In PROLOG these statements would be written as:

```
sheepdog(collie);                fact that a collie is a sheepdog
hound(beagle)          fact that a beagle is a hound

passed a ball(George,Steven);
```

A rule contains a condition:

```
Cats purr if they are stroked by humans.
```

This would be expressed in PROLOG as:

`purr(X) :- stroked by(X,humans)` where `X` = cats

Facts and rules form the basis of PROLOG programs to represent knowledge that are stored in a database. This database can then undergo querying.

Example - Using facts, rules and queries

Problem: Suppose we want to find out whether a person drives a fast car. We start by building a set of facts and rules for our knowledge database.

Solution:

```
person(judy);  -  this is the fact that Judy is a person
person(james);
drives_car(james,ford escort);
drives_car(judy,porsche); - this is the fact that Judy drives a Porsche
drives_fast_car(X) :- drives_car(X,Y)
                      and Y = "porsche"- this is a rule for X driving a fast car.
```

In this example we could ask the program to tell us whether Judy drives a fast car by typing the goal:

`?drives_fast_car(judy)`. The result would be YES since the goal is satisfied.

If we asked:

`?drives_fast_car(james)`

then the result would be NO as `drives_car(james,Y)` would evaluate `Y="ford escort"`.

This would then cause the rule `drives_fast_car(james)` to fail as `Y` does not equal "porsche" and the goal is not satisfied.

You can see from the code that there is no description of the type of data or its internal representation. There are simply statements of facts and a rule.

Contrast this with a procedural language where the programmer would need to set up a structure to hold the knowledge and predefine its type (string, number etc). Then they would need to describe the steps taken to search the structure in order to answer the query. A declarative/logical language is simplistically described as telling the computer *what to do* and *not how to* do it.

Goals and clauses

On the Web is a interactivity. You should now complete this task.

5.6 Event-driven languages

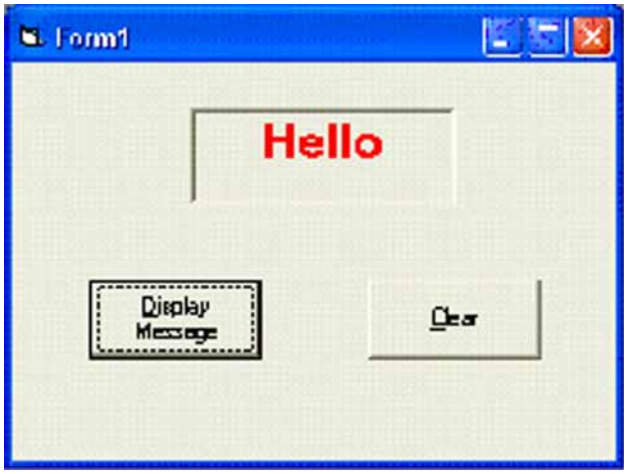
Event driven programming languages have evolved to handle events.

Events can be initiated at two levels:

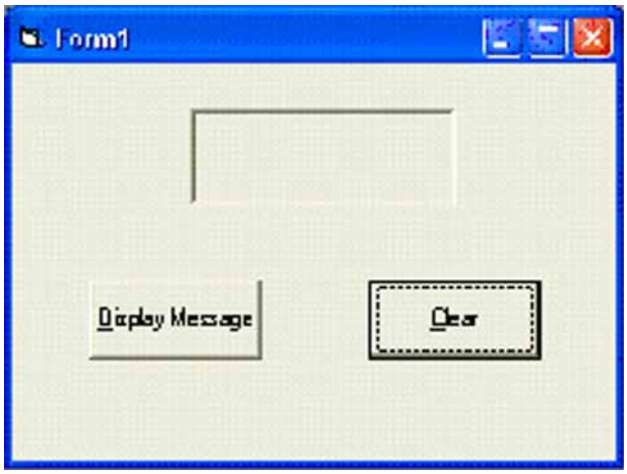
1. at *system/hardware* level: events can include timers, interrupts, loading of files etc;
2. at *language/system* level: events involve mouse clicks, keyboard presses and cursor movements.

In Visual Basic each object has a set of events. Here an event is an action that Visual Basic can detect and respond to by a user clicking on a command button, for example.

The following examples show the action of clicking on two buttons in a Visual Basic program:



Button1 'Display Message' produces a message in the label window



Pressing the 'clear' button removes the message.

A typical event driven program has the effective structure:

```
do (forever)
  if event then
    if event caused by X then
      handle X
    else if event caused by Y then
      handle Y
```

After each event is handled, nothing happens until the next event occurs.

Note that event-driven programs do not have a predefined pathway in the execution of the code, as opposed to imperative programming style i.e. they have no beginning or end.

Graphical user interface programs are typically programmed in an event-driven style using languages such as Visual Basic and Visual C.

Even Java, an object-oriented language can be used for event-driven Windows-style programming with AWT (*Abstract Window Toolkit*). This a large collection of resources for building graphical user interfaces within the Java environment.

Sentence completion - languages

On the Web is a interactivity. You should now complete this task.

5.7 Scripting languages

A **Scripting language** is a style of 'programming' that produces ASCII text-based scripts which are usually designed for writing small programs like batch files. Scripting languages support high level language control features such as selection and iteration (syntax) but they are not considered programming languages as such. Often referred to as '*glue-code*' they are instead, seen as being an enhancement of particular software packages.

Scripting languages are not new! *Job Control Language (JCL)* was one of the earliest scripting languages to be used in the 1960s and 1970's on mainframe computers using punched card input. The main program cards were preceded by cards containing JCL code that initialised system requirements such as memory, buffers etc and formatted the input and output parameters.

For example the following code requests buffer memory and formats the input and output data streams for all subsequent program runs:

```
BUFFERS 64
FILE CARDS "INPUT" (2,10), LINES 1 (2,17)
```

Examples of present day scripting languages are *VBScript, JavaScript, Perl, Python, TCL (Tool Command Language)* etc

Applications that provide scripting capability allows the user to extend the functionality of the application by programming a sequence of actions. For example, in *Filemaker Pro* (a database package) it is possible to write scripts that open and close files, copy data from records or enter a certain database mode such as *browse* or *find*.

An example script in *Filemaker Pro* is shown in Code 5.1

```
Enter Browse Mode[]
Go to Layout ["Layout 1"]
Enter Find Mode []
Set Field ["Computing Course", ""S3C8""]
Perform Find[]
Sort [Restore, No Dialog]
Go to Layout["S3 Report Final"]
```

Code 5.1

Another example is VBScript which is a cut-down version of Visual Basic, used to enhance the features of web pages in Internet Explorer. Below in Code 5.2 is an example of a section of VBScript embedded in HTML code.

```
<HTML>
<HEAD>
<TITLE> This is VBScript in action!</TITLE>
<SCRIPT LANGUAGE="VBScript">
MsgBox "Welcome to VBScript"
<P> Click on the button below </P>
<INPUT TYPE ="Button" NAME="cmdClick" VALUE="Click">
</SCRIPT>
```

Code 5.2

Note that the scripting language must be specified in the browser.

5.7.1 Benefits of scripting languages

One of the main benefits of scripted languages is that they require no compilation. The language is interpreted at run-time so the instructions are executed immediately.

Scripting languages also have a simple syntax which, for the user:

- makes them easy to learn and use
- assumes minimum programming knowledge or experience
- allows complex tasks to be performed in relatively few steps
- allows simple creation and editing in a variety of text editors
- allows the addition of dynamic and interactive activities to web pages

Also, scripting languages are generally portable across various hardware and network platforms and scripts can be embedded in HTML documents for added functionality.

Specialised scripting languages include:

Perl (Practical Extraction and Report Language). This is a popular string processing language for writing small scripts for system administrators and web site maintainers. Much web development is now done using Perl.

Hypertalk is another example. It is the underlying scripting language of HyperCard, while Lingo is the scripting language of *Macromedia Director*, an authoring system for develop high-performance multimedia content and applications for CDs, DVDs and the Internet.

AppleScript, a scripting language for the Macintosh allows the user to send commands to the operating system to, for example open applications, carry out complex data operations. An example script is shown in Code 5.3

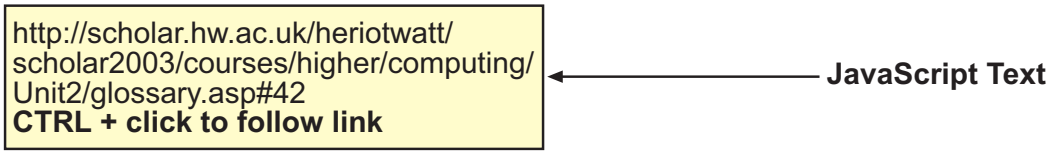
```
--this simple Applescript will search
--recursively through folders in a shared volume
--removing certain unneeded files from every user's
--directory

on iterate(thisFolder)
 tell application "Finder
  if the (count of folders in thisFolder) > 0 then
   repeat with innerFolder in every folder of thisFolder as list
    if file "Mail Drop Preferences" of innerFolder exists then
     delete file "Mail Drop Preferences" of innerFolder
    end if
    if file "Email Startup" of innerFolder exists then
     delete file "Email Startup" of innerFolder
    end if
    if file "EmailSavePreferences" of innerFolder exists then
     delete file "EmailSavePreferences" of innerFolder
    end if
   end repeat
  end if
  end tell
end iterate
```

Code 5.3

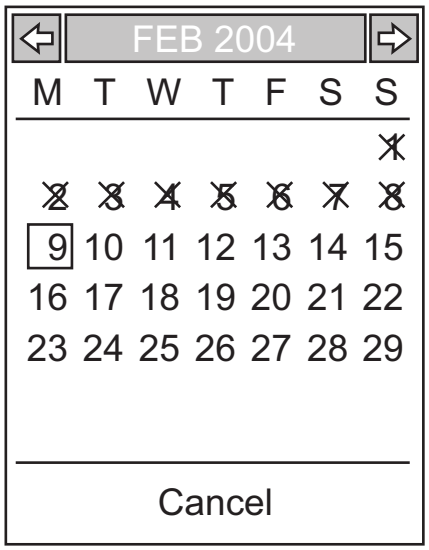
JavaScript, perhaps the most publicised and well-known scripting language was initially developed by Netscape as LiveScript to allow more functionality and enhancement to web page authoring that raw HTML could not accommodate. A standard version of JavaScript was later developed to work in both Netscape and Microsoft's Internet Explorer, thus making the language to a large extent, universal. This means that JavaScript code can run on any platform that has a JavaScript interpreter.

Typical uses of JavaScript include:

a) **Image or text rollovers.** If the user rolls the mouse over a graphic or hypertext then a text or graphic box will appear:

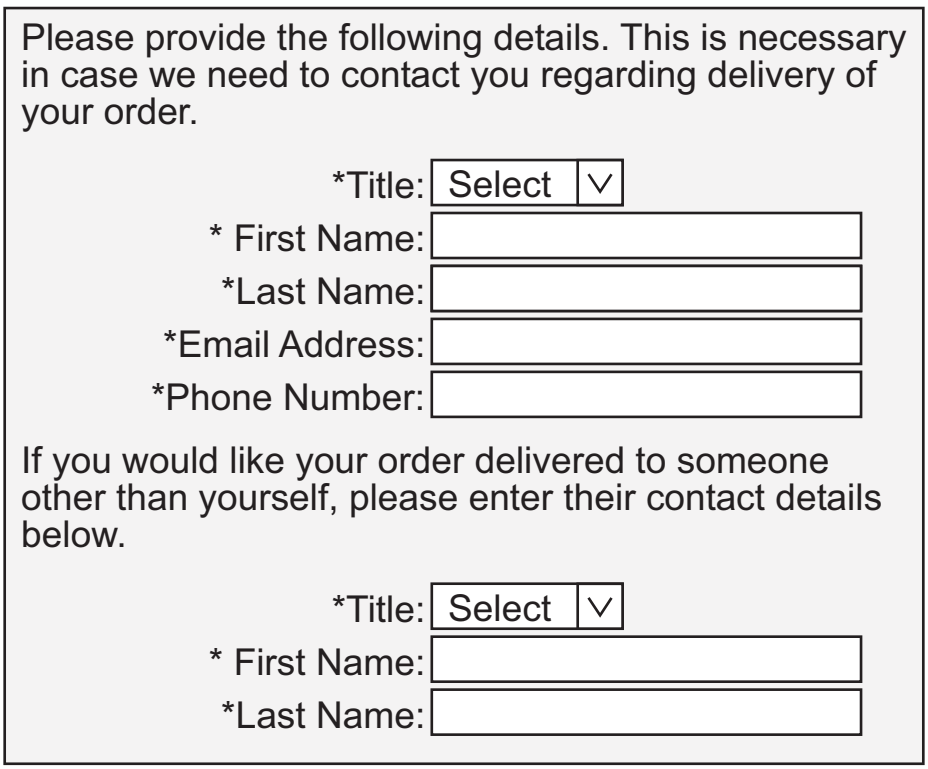


b) **Creating a pop-up window** to display information in a separate window from the Web page that triggered it. This is useful if the user requires to perform a simple calculation or consult a calendar for inputting dates. This is achieved by embedding ActiveX controls or Java applets into the script.



Pop-up Calander

c) **Validating the content of fields.** When filling in forms, each field, especially required fields denoted by an asterisk, are validated for correct input. If the field is left blank or incorrect information entered then a user message will be generated and you may not continue.



5.7.2 The need for scripting languages

Nowadays scripting languages are becoming more popular due to the emergence of web-based applications. The market for producing dynamic web content is now expanding extremely rapidly such that new scripting languages have been developed to allow users with little or no programming expertise to develop interactive web pages with minimum effort.

Also the increases in computer performance over the past few years has promoted a comparable increase in the power and sophistication of scripting languages that, unlike conventional programming languages, can even have certain security features built-in. Downloading web-based content from a remote site to a user's local machine can include animations, graphics, MP3 audio files, video clips and so on and this is authenticated by the scripting language.

However be warned! Executable code can inadvertently be downloaded from a remote server to a web browser's machine, installed and run using the local browser's interpreter. This is easily done by visiting dubious web sites or downloading programs without valid authenticity. The user is probably unaware of anything devious occurring. This is a weakness in the formal rules defining scripting languages like JavaScript and VBScript.

5.7.3 Creating a Macro

A **macro** is a way to automate a task that you perform repeatedly or on a regular basis. It is a series of commands and actions that can be stored and run whenever you need to perform the task. Instructions can be simple, such as entering text and formatting it, or complex, like automating tasks that would take several minutes to do manually Macro contents consist of ASCII text and can be created and edited in any simple text editor.

Many programs (like Microsoft Word and Microsoft Excel) can create macros easily. All you have to do is "record" a set of actions as you perform them. For example, you could record opening a new document using a specific template, inserting a header and inserting a name and address and greeting. Each time you "replayed" the macro, it would perform those tasks.

For programs that don't include a macro facility there are numerous shareware macro programs that can be downloaded and used in any application. Such a program is *Macro Express 2.1* which is ideal for beginners.

5.7.4 Running A Macro

A macro can be initiated by:

- pressing selected key combination (hot keys)
- clicking an icon on the toolbar that has been created for the macro
- running the macro from the Tools menu of the application.

Example tasks could include:

- inserting your name and address on documents
- formatting text with specified font and size
- accessing websites from a list of 'favourites'
- inserting special symbols or graphics into documents
- automate playing of audio CDs while you work on the computer

- formatting of spreadsheet cells
- creating headers and footers.

Code 5.4 is an example VBScript listing for a macro to create a table in Microsoft Word:

```
'Macro to create a table in Word
'
Sub Macro1()
'
' Macro1 Macro
' Macro recorded 05/01/2004 by John Smith
'
    ActiveDocument.Tables.Add Range:=Selection.Range, NumRows:=7,
    NumColumns:= _
        4, DefaultTableBehavior:=wdWord9TableBehavior,
    AutoFitBehavior:= _
        wdAutoFitWindow
    With Selection.Tables(1)
        If .Style <> "Table Grid" Then
           .Style = "Table Grid"
        End If
        .ApplyStyleHeadingRows = True
        .ApplyStyleLastRow = True
        .ApplyStyleFirstColumn = True
        .ApplyStyleLastColumn = True
    End With
End Sub
```

Code 5.4

5.7.5 Review Questions

Q11: An event-driven language differs from other languages in that (choose one):

a) Programs have no pre-defined pathway
b) Programs are initiated entirely from mouse clicks
c) Program events cannot be nested
d) Programs are difficult to write

Q12: One of the main benefits of a scripting language is:

a) They are easy to learn and use
b) Allow complex tasks to be performed in relatively few steps
c) Web pages can be made more dynamic
d) All of the above

Q13: The need for scripting languages has emerged from (choose one):

a) An increase in the speed of the Internet
b) Ease of developing of interactive web pages
c) The creation of JavaScript
d) All of the above

Q14: Many software applications include a macro facility. Which one of the options best describes a **macro**?

a) Macros are not easy to edit
b) The creation of a macro can be time-consuming
c) Macros automate repetitive tasks
d) A macro listing consists of complex code

Q15: One of the tasks that a macro could NOT perform would be:

a) Inserting special symbols or graphics into documents
b) Formatting a floppy disc
c) Creating headers and footers
d) Changing audio CDs

5.8 Object-oriented languages

Object-oriented programming was borne out of the need to overcome the problems of conventional programming projects that could involve many thousands of lines of code; the latest Window OS, for example contains many *millions* of lines of code. Needless to say managing projects of this magnitude are extremely error-prone; changing the program specification or tinkering with a few lines of code can mean a complete re-write of program modules probably involving several month's work.

Object-oriented programming (OOP) requires a new way of thinking that might seem strange to you at first sight. Object-oriented programming is a programming language model organized around *objects* rather than *actions* and data rather than logic. Object-oriented programming takes the view that what we really care about are the objects we want to manipulate rather than the logic required to manipulate them.

The primary concept in object-oriented programming languages is a *class*. A class is a user-defined type that represents an object or a concept-a file, a hardware device, a database record, a document, an airplane, a chair, a computer etc. in fact anything in the real world.

An *object* is an instance of a class, just like a variable in procedural programming languages is an instance of a type. An object is an entity in itself that can interact with other objects. The first step in object-oriented programming is to identify all the objects you want to manipulate and how they relate to each other, an exercise often known as **data modelling**. Once you've identified an object, you generalize it as a class of objects

For example there may be a class Cat and the cat Kitty is an object - she is an instance of the class Cat. The objects are created and destroyed but the class continues to exist. (For example, Kitty may die but the class of Cat still exists; or I could get another cat called Puss and then have 2 objects both of type cat.) The class is the data type and

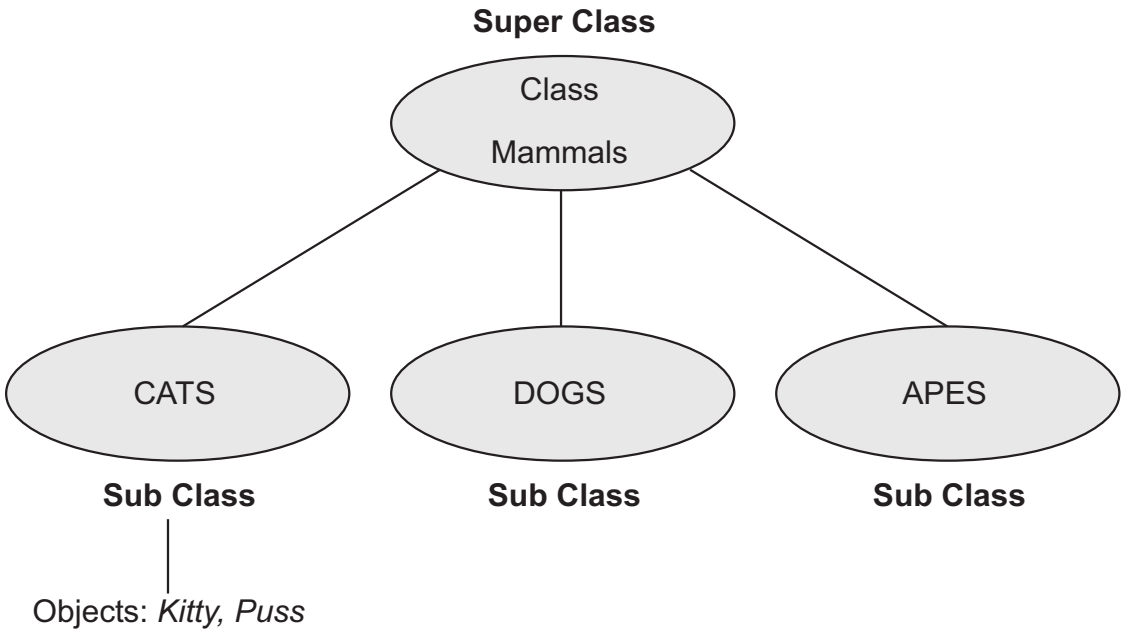
the object is the variable

One of the main benefits of object-oriented programming is the creation of *subclasses* of a class that share some or all of the main class characteristics. Called **inheritance** the subclass then *inherits* all the properties and methods of the *superclass*.

In this way, inheritance can promote code reuse: instead of recreating all the code common to both classes, you can simply extend an existing class. Another subclass, in turn, could extend the Cat class, and so on. In a complex application, determining how to structure the hierarchy of your classes is a large part of the design process.

For example, a Mammal class could be created that defines the properties and behaviours common to all mammals. The Cat class would then be an extension of the Mammal class with similar traits.

This is shown diagrammatically:



This property of object oriented programming forces a more thorough data analysis, reduces development time, and ensures more accurate coding

Java

One of the first **object-oriented** computer languages was called Smalltalk. C++ and Java are the most popular object-oriented languages today. In fact Java is an example of a language that is platform independent. The Java compiler produces a sort of machine code called **bytecode**. This can't run directly on any machine. Rather it runs the Java virtual environment. This is a piece of software that is specially designed for the architecture of the computer in question. This is freely available on the internet and enables the computer to run Java programs. A Java-enabled computer, whatever the architecture, can run bytecode i.e if Java is written on a Sun workstation, the program will run on PCs and Apple Macs.

Object orientation

On the Web is a interactivity. You should now complete this task.

Derived classes

On the Web is a interactivity. You should now complete this task.

Inheritance

On the Web is a interactivity. You should now complete this task.

5.9 Functional languages

Functional programming emphasizes the evaluation of expressions rather than execution of commands. The expressions in such languages are formed by using functions to compute values. Also from the collection of primitive functions new functions can be defined and created.

Consider, for example, the simple problem of calculating the sum of the integers from 1 to 100. A simple loop structure could be used as seen in the following pseudo-code:

```
1 Set total to 0
2 Set counter to 0
3 Repeat
4  counter = counter + 1
5 total = total + 1
6 Until counter = 100
```

Using a **functional language** to solve the same problem, the result can be calculated by evaluating the one-line expression, without the use of any variables:

```
sum [1..100]
```

In this case the expression [1..100] represents the list of integers from 1 to 100 while sum is a function that can be used to calculate the addition of the integers.

Lisp

The first functional language was Lisp, which is a *LISt-Processing* language. A list is a data structure, consisting of structured values (arrays and records for example), whereas the basic data in imperative languages are essentially unstructured (integers, for example).

Lisp is in fact one of the oldest programming languages, older than any imperative language except Fortran, and was first developed in the late 50s. It has widespread use in the field of artificial intelligence.

5.9.1 Review Questions

Q16: One example of an object-oriented language is:

a) Prolog
b) Visual Basic
c) Java
d) COBOL

Q17: The main concept in an object-oriented language is class. Which one of the following represents a **class**?

a) A word processing document
b) A printer that is on-line
c) A database record
d) All of these

Q18: Object-oriented programming can involve inheritance. Which one of the following best describes what is meant by **inheritance**?

a) Data is passed from one part of the program to another
b) The behaviour of a class object cannot be changed
c) It can promote code re-use by extending an existing class
d) It focuses mainly through data structures

Q19: One of the earliest known functional languages was:

a) Lisp
b) Fortran
c) Visual C++
d) Java

Q20: A functional language differentiates itself from other programming languages in that (choose one):

a) It focuses on the execution of commands
b) It is an easy language to learn
c) It emphasises the evaluation of expressions
d) It produces largely unstructured code

5.10 Translation methods

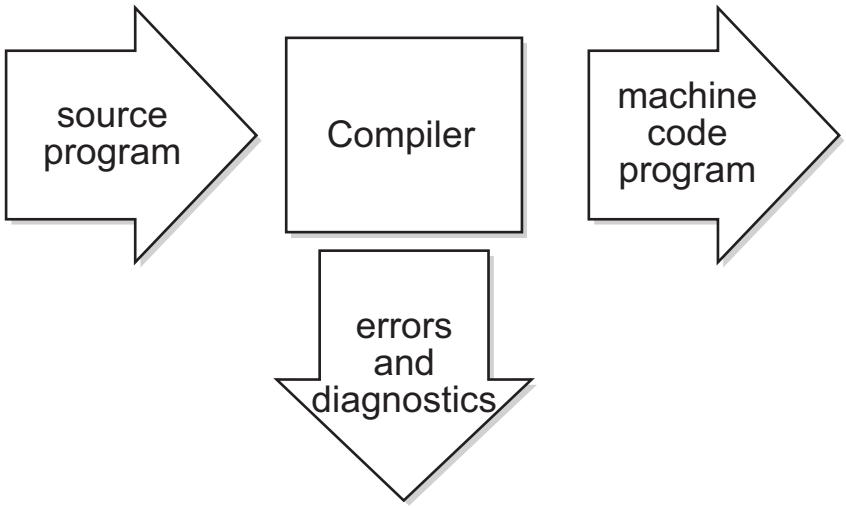
At the end of the implementation stage, all going well a structured program listing will be produced, complete with internal documentation. This will be thoroughly checked against the design and against the original specification.

The high-level code written at this stage is called source code which must be translated into machine code, called object code that the computer understands.

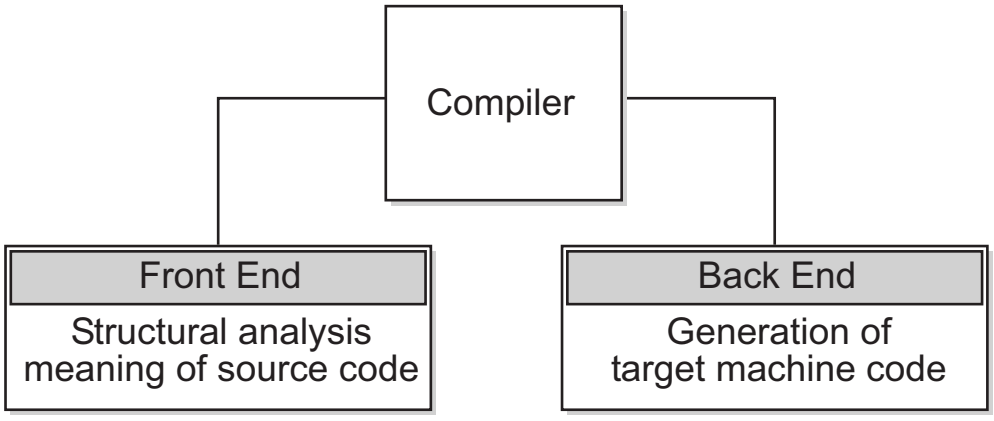
There are two methods of translating source code into object code; a compiler and an interpreter.

5.10.1 Compiler

A compiler, which is a complex program in itself, translates source code into object code that is then loaded into main memory and executed.



Translating source code into object code is a complicated task that it is unreasonable to expect it to occur in a single, coherent step Compilation is thus broken down into several logical *phases* as seen in the following diagram:



At the highest level, a compiler consists of a *front end* and a *back end*. The front end carries out an analysis of the structure (**syntax**) and meaning (**semantics**) of the source code, while the back end generates the object code.

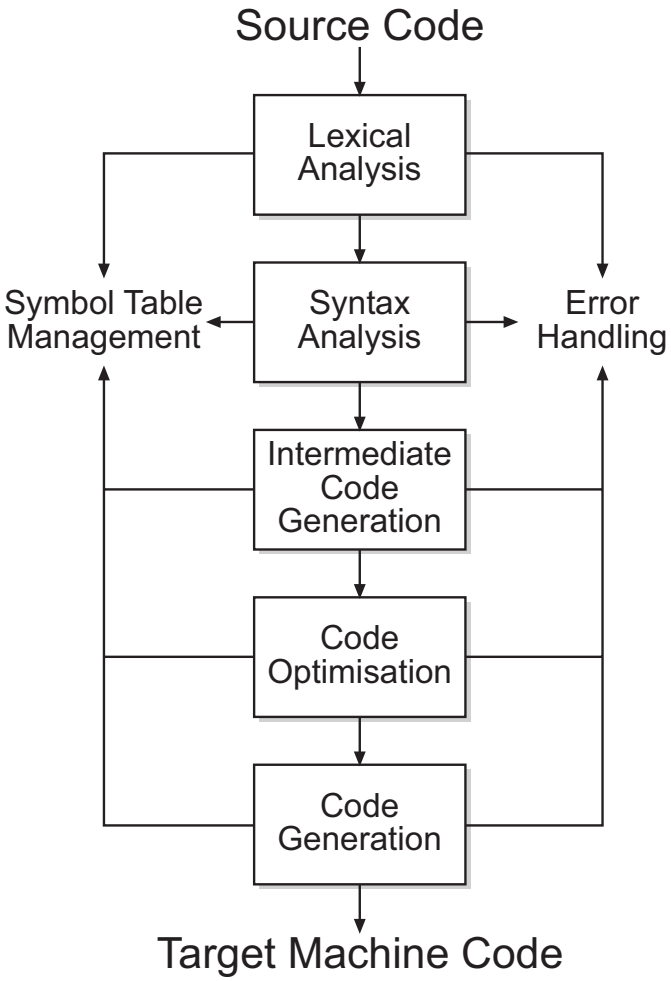
Looking at the operations of the front end in a little more detail it can be seen as being composed of the following three stages:

1. lexical analysis
2. syntax analysis
3. semantic analysis

A closer look at the operations of the back end reveals:

1. intermediate code generation
2. code optimisation
3. machine code generation

The sequencing of these phases is shown in the following figure:



Lexical analysis takes the source program and constructs source program words and symbols (invariably called tokens) for all the identifiers, reserved words like PRINT and arithmetic operators. All comments and superfluous spaces are removed and the tokens stored in the **symbol table**.

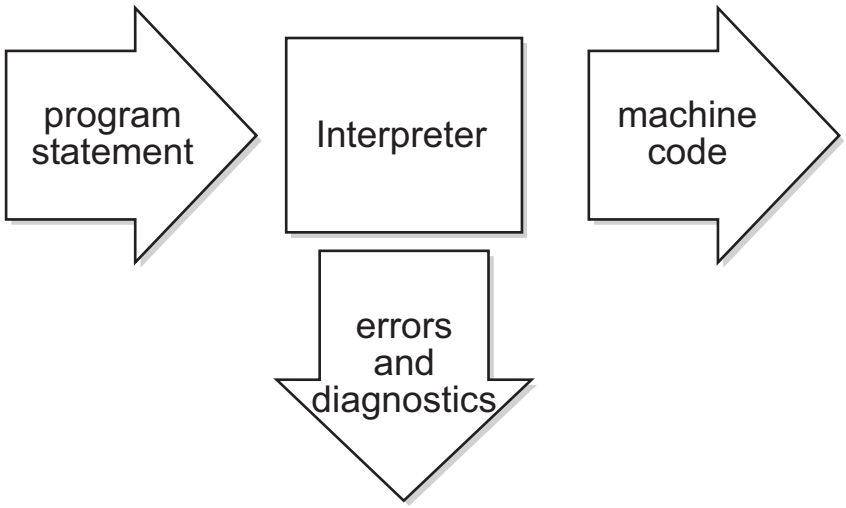
Syntax analysis checks the symbol table contents to ensure that there are no *grammatical* or *semantic* errors in the source code. The type of error will depend upon the syntax definition of the programming language. For example, some languages require semi-colons to mark the end of statements, blocks of code are defined within a BEGIN END, etc. Some compilers are also able to check that the correct number of parameters is being passed in the call to a subroutine and that they are of the appropriate data type.

If any errors are detected then the compiler will pass control to the error-handling routine and generate an *error listing* file containing descriptions of the offending statements and their line numbers within the source. The programmers will then check through the error listing and use this as a basis for debugging the original source code. The source code must then be recompiled. Only when there are no errors will an object code file be generated for execution.

Intermediate code is produced for the target computer and this is then **optimised** to produce more efficient code that might run faster if part of a network, for example. Once optimised the final object code is generated which can then be run or executed.

5.10.2 Interpreter

Another form of translation that converts source code into object code is an interpreter.



Unlike a compiler, an interpreter checks syntax and generates object code one source line at a time. Think of this as very similar to a group of translators at a United Nations' Conference, who each have to convert sentences spoken by delegates into the native language of their representative.

When an error is encountered, the interpreter immediately feeds back information on the type of error and stops interpreting the code. This allows the programmer to see instantly the nature of the error and where it has occurred. He or she can then make the necessary changes to the source code and have it re-interpreted.

As the interpreter executes each line of code at a time the programmer is able to see the results of their programs immediately which can also help with debugging.

5.10.3 Respective Advantages

The main difference between an interpreter and a compiler is that compilation requires analysis and the generation of machine code only once, whereas an interpreter may need to analyse and interpret the same program statements each time it meets them e.g. instructions appearing within a loop.

For example the following Visual Basic code would be interpreted each time the loop is entered:

```
For iCountvar = To 20
iSum = iSum + iCountvar
Pic.Display iSum
Next iCountvar
```

Errors

This has implications for error reporting. For instance, when the interpreter encounters an error it reports this to the user immediately and halts further execution of the program. Such instant feedback, pinpointing the exact location of the error, helps the programmer to find and remove errors.

Compilers, on the other hand, analyse the entire program, taking note of where errors have occurred, and places these in an error/diagnostic file. If errors have occurred then the program cannot run. Programmers must then use the error messages to identify and remove the errors in the source code.

Some compilers assist by adding line numbers to the source listing to help pinpoint errors and all compilers will describe the nature of the error e.g. missing semi-colon, expected keyword, etc. - although interpreting some compiler diagnostics is a skill in itself.

Error correction can be very time-consuming and frustrating, particularly in the case where spurious errors occur, e.g. many errors are highlighted in the source but the cause of the error is a single, simple mistake. An example of this would be errors that are generated by, say, a compiler, if a programmer simply misses out a semi-colon.

Speed

Another important difference is that interpreters can be 2 to 10 times slower than compilers. One reason for this is that they translate the same statements within a loop over and over again.

Compilers can produce much more efficient object code than interpreters thus making the compiled programs to run faster.

Ease of use

Interpreters however are more suitable for beginners to programming since errors are immediately displayed, corrected by the user, until the program is able to be executed.

On the whole compilers tend to be more difficult to use.

5.10.4 Review Questions

Q21: One of the main functions of a compiler in translating high level languages is to produce a symbol table containing reserved words and identifiers. This part of the process is called:

a) Program analysis
b) Lexical analysis
c) Semantic analysis
d) Syntax analysis

Q22: Syntax analyses is rather more than just finding terms that are mis-spelled. In a computer program which one of the following will also be found?

a) Missing parenthesis
b) Use of a reserved word
c) Inappropriate data type
d) All of these

Q23: One of the main differences between a compiler and interpreter is:

a) An interpreter is faster then a compiler
b) A compiler is better for beginners
c) A compiler can produce more efficient object code
d) An interpreter is much harder to use

Q24: One disadvantage of using an interpreter is:

a) Looping structures have to be interpreted each time they are entered
b) It stops execution when an error is encountered
c) It helps the user to debug programs
d) An interpreter is ideal for beginners

Q25: High level languages have to be translated because (choose one):

a) Computers can only understand machine code
b) Source code is faster to run than object code
c) Programs run faster when converted to binary
d) All of the above

5.11 Summary

The following summary points are related to the learning objectives in the topic introduction:

- there are numerous types of programming languages in use today;
- they are difficult to organise into discrete categories because of overlapping properties;
- features of procedural languages lend themselves to most of the programming tasks of the Higher computing course;
- high level languages have to be translated to machine code by compiler or interpreter;
- syntax and semantics of the language are part of the translation process;
- types of errors can be picked up by translators;
- scripting languages are now of great use in web page design.

5.12 End of topic test

An online assessment is provided to help you review this topic.

Topic 6

High Level Language Constructs 1

Contents

Prerequisite knowledge

Before studying this topic you should be able to describe and use the following constructs in pseudo-code and a suitable high level language:

- *input and output;*
- *numeric and string variables;*
- *assignment statements;*
- *arithmetical operations (+, -, *, /, ^);*
- *logical operators (AND, OR, NOT)*
- *conditional loops using fixed and complex conditions;*
- *IF statement;*
- *nested loops.*

Learning Objectives

- *understand the need for programming variables*
- *understand and be able to use real, integer, string and boolean variables*
- *understand string operations such as concatenation and substrings*
- *be aware of local and global variables*
- *understand what is meant by the scope of variables*
- *understand the nature and use of sub procedures*
- *understand the use of selection in programming an in particular the CASE construct*

Revision

Q1: Which one of the following program constructs does NOT represents the process of selection?

a) The IF..THEN statement
b) The FOR..NEXT loop
c) The CASE statement
d) The DO..WHILE loop

Q2: One of the following program statements produces the value 13 for the variable Answer. Which one?

a) Answer = 5 + 8 * 3 - 2
b) Answer = (5 + 8) * 3 - 2
c) Answer = 5 + (8 * 3) - 2
d) Answer = 5 + 8 * (3 - 2)

Q3: If the value of 1 represents TRUE and 0 represents FALSE which one of the following statements is true?

a) 1 AND 0 = TRUE
b) NOT 1 = FALSE
c) 1 OR 1 = FALSE
d) 0 AND 0 = TRUE

Q4: Consider the following segment of programming code:

```
For x = 1 to 3
For y = 1 to 3
Sum = x + y
Next y
Next x
Print Sum
```

When run what would be the final value of the variable Sum?

a) 2
b) 4
c) 6
d) 8

Q5: Which one of the following statements represents a valid string assignment?

a) Name = "Bert"
b) Number = "12345"
c) Paper = "Scots" + "man"
d) All of the above

6.1 Introduction

In this topic you will be introduced to the Visual Basic programming environment. Variables and data types are discussed with reference to local and global forms and scope, together with fundamental programming statement structures. Formatted input and output constructs are described and exemplified using respective Visual Basic code. More advanced techniques involve conditional statements such as nested IF statements and the CASE construct. Each Visual Basic construct is exemplified using pseudo-code and high level equivalent. .

6.2 The Programming Environment

In this topic, and others all language features are exemplified using Visual Basic 6.

Users of Visual Basic.NET should find few incompatible problems. Important differences will be flagged, where appropriate.

Visual Basic 6

Visual Basic is an events driven programming language that works within the graphical environment of Windows. Visual Basic code is associated with objects and each object has a set of events associated with it.

Events are actions which Visual Basic detects and respond to. For example, a user clicking on a command button on a form will generate a click event for that button; pressing a key on the keyboard could initiate a load event for a programming module. When an event is generated Visual Basic will run any code that you have entered for that event. Since there may be many objects associated with a single program it means that each object must be coded separately.

Objects are coded in Visual Basic using sub procedures that may be executed individually or be linked together in one way or another.

Visual Basic Environment

When Visual Basic is launched using the Standard EXE option a number of separate windows appear as seen in Figure 6.1

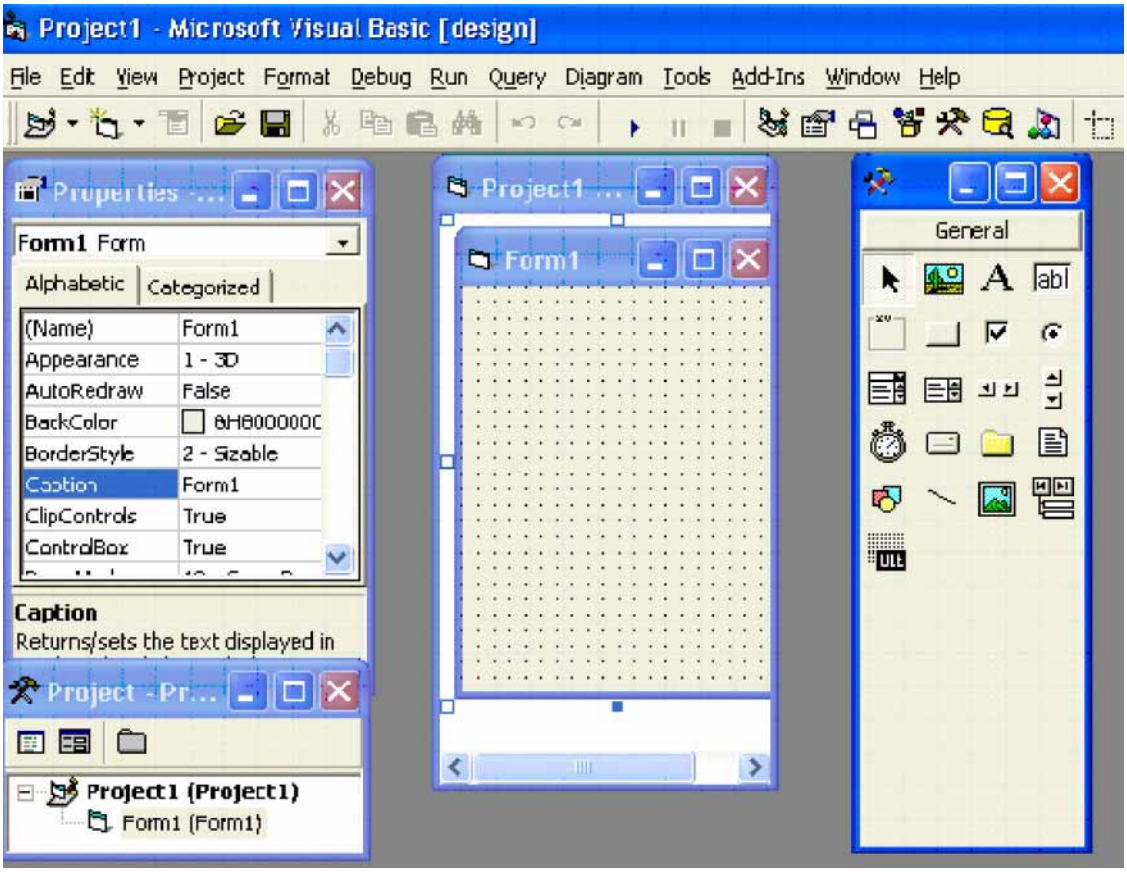


Figure 6.1:

The default screen that you can see is in design mode and consists of four main windows:

1. a blank Form window which is the interface with the application (program) you create. The visual design of the program is created on this form, which has gridlines to help build objects like text boxes, label boxes and control buttons etc. What is placed on a form will be seen in a window when the program is executed.
2. a Project window that displays the files that are created during the construction of the program. These files could be forms, modules (blocks of code not attached to a form), graphics, or control structures such as Active X, required for the successful running of your program. Note that only one project can be open at any given time.
3. a Properties window that displays the properties of the objects created in the program. The form window itself has properties associated with it.
4. a Toolbox window that consists of all the controls necessary for developing a program. Boxes, labels, buttons, and other objects can be drawn on the form as part of the visual interface and also to allow input and output of data.

At this stage it is important to realise that when you create a program, each form, module, graphic, and ActiveX control is saved as an individual file

Table 6.1 shows the common files types in a Visual Basic Project:

Table 6.1:

File Type	Description
FRM	Form
BAS	Module
OCX	ActiveX control
CLS	Class module
VBP	Visual Basic project

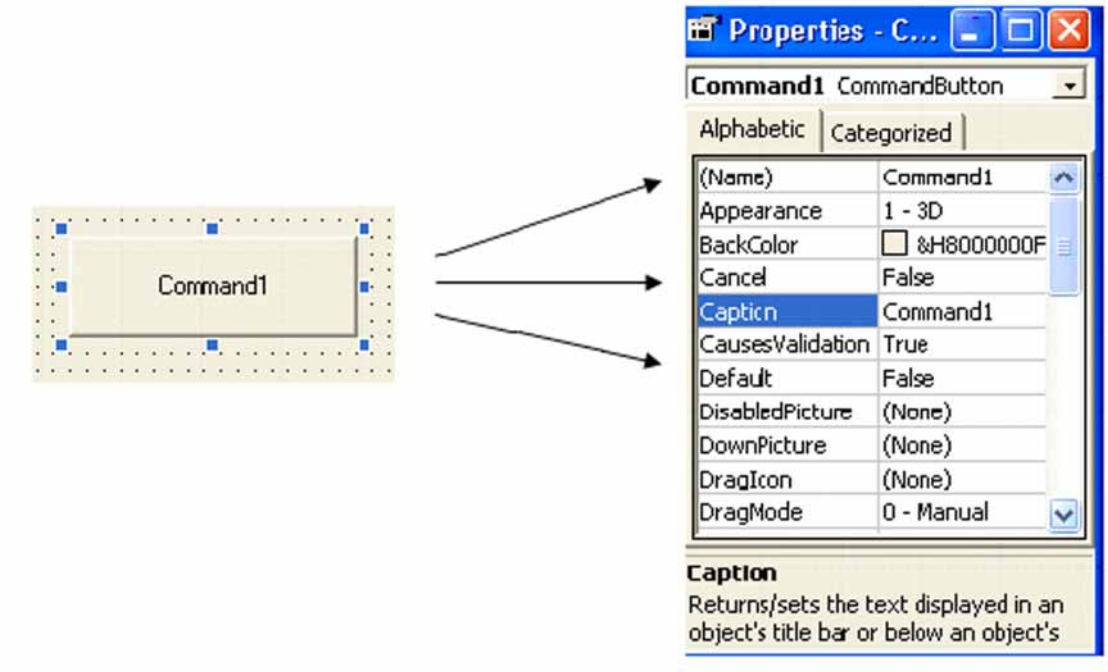
When starting to write Visual basic programs the two most common file types are *Form* and *Project*.

6.3 Building applications

Constructing a program in Visual Basic involves two processes:

1. creating the visual design of the program
2. entering and implementing programming code.

In the first process the properties of the form are established followed by the creation of controls inside the form. The properties of each control are then established.



Pressing function key F4 on an active button will open the Properties box as seen above.

Figure 6.2 shows five objects: 3 control buttons, a message box, and a form each with event coding.

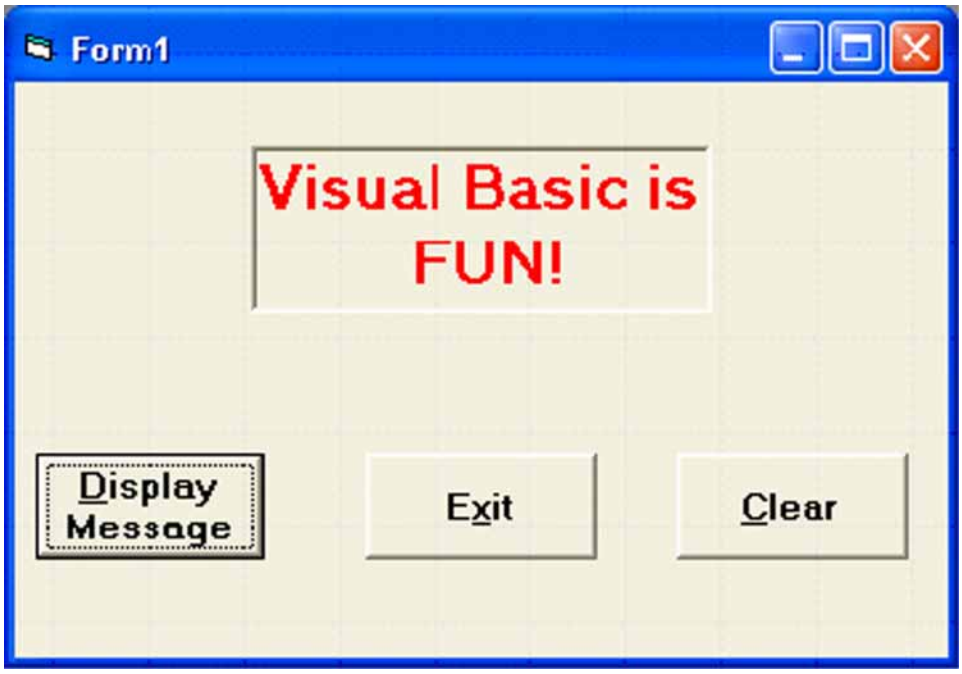


Figure 6.2:

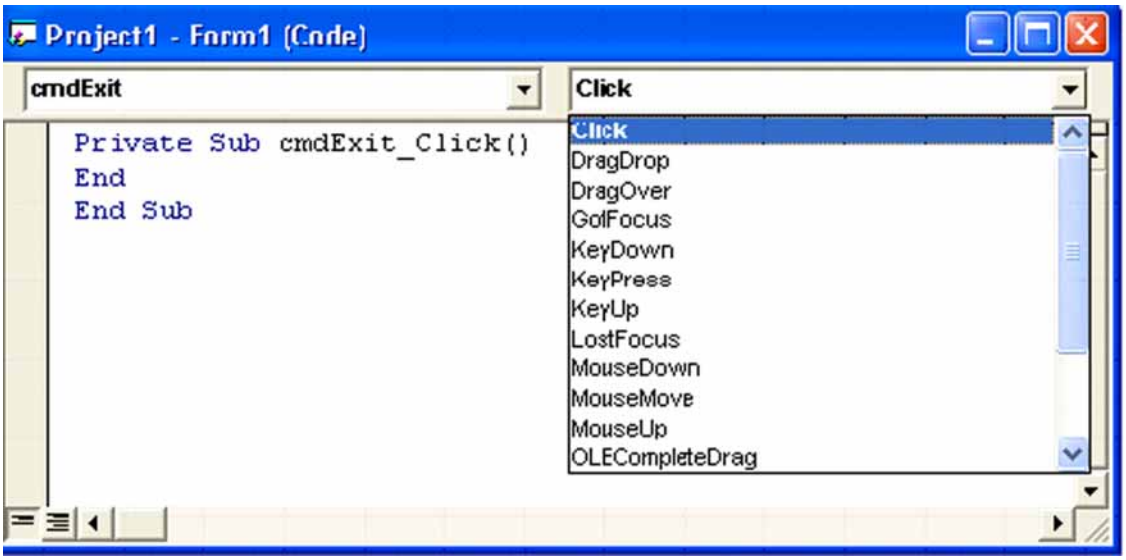
In the second process programming code is written for each event and attached to each object in the form.

The code for each object is written in the form of a sub procedure.

A sub procedure or sub program is a block of code that is implemented when an event, such as a mouse click is actioned. Sub procedures can be used anywhere in the program by *calling* them by name. Procedures can also be user-defined.

More on procedures will be covered later in the next topic.

Double-clicking on any of the objects in a form or even the form itself will open up the program editor and reveal the code for each event. For example clicking on the *Exit* button will display the code editor as follows:



All Visual Basic code is written between the lines that Visual Basic enters for you:

```
Private Sub cmdExit_Click()
End Sub
```

On the right of the screen you can see a list of other events that are part of the Visual Basic environment.

6.3.1 Input and Output Controls

Visual Basic programs do not output to the computer screen as such. Instead they use a *form* which is, essentially a Visual Basic window that is the interface between the user and the application and allows information to be input by the user or displayed to the user. There may be more than one *form* in a Visual Basic project.

Information may be input to a program via the keyboard or read directly from data files. The program may also contain data that is assigned directly to variables or constants during execution time.

The output of information may be achieved through a variety of Visual Basic control structures that are placed on the form at the design stage. Output can also be directed to a printer or to an existing file.

Since file I/O is beyond the scope of the Higher Computing course, only the graphical methods will be discussed at this stage.

The stages in writing a program in Visual Basic involve the following aspects:

- **Design stage**: the form is used to compose the graphical elements of the program. Command buttons and control boxes (objects) can be placed anywhere on the form and these will dictate how the application is to run.
- **Setting properties**: the form itself and the objects it contains can have various properties set thereby producing the visual effects required of the program. Although there are numerous properties for each object the main ones would include:

Name of the object;

Colour;

Caption attached to object

Height/width

Font used;

Position of object within the form;

Border style.

Properties are changed by highlighting the object and pressing function key F4. This opens up the Properties window from which changes can be made.

- **Coding stage**: code is written for each object and when the main program is run each event will become part of the overall effect. Example events would include:

  Clicking a command button;

  Loading a form;

  Clearing the contents of a text box;

  Ending a program.

When Visual Basic is started, *Form1* shows by default.

You will see exemplar programs that make use of the following Visual Basic constructs later in this topic and others.

Input methods

1 InputBox function

A program may ask for user input during its execution. This can be accomplished using the InputBox function, which by default allows text entry.

The full syntax for the InputBox function is:

```
InputBox "Prompt", "Title", xpos, ypos
```

`Prompt`: is the displayed message

`Title`: is the optional text that will appear in the title bar

`xpos,ypos`: coordinates for positioning the InputBox on the screen.

Note that in Visual Basic, the screen is measured in Twips (Twentieth of a point). A point is a traditional measure used in printing which measures approximately 1/72nd of an inch or 1/28th cm.

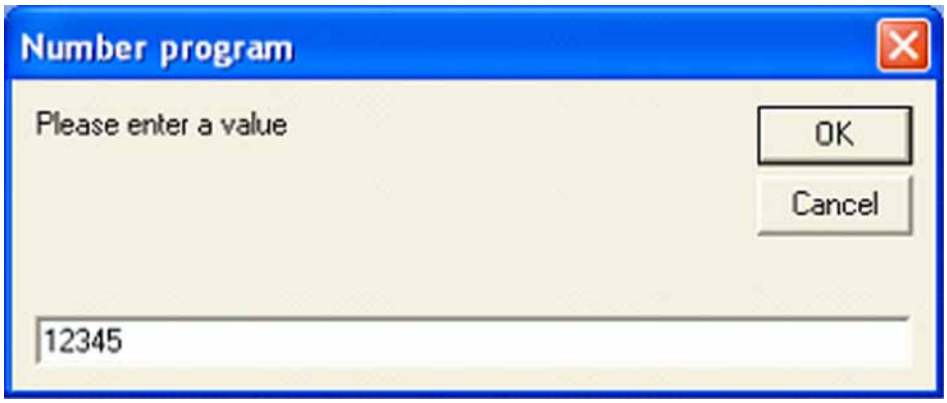
For example a program may require the user to enter a name. The format is:

```
Name = InputBox("Please enter a name",)
```

The user then complies with the request to enter a name.

Should a numerical value be required by the program then the input format is identical and the text input converted to numeric using the VAL function

```
Number1 = Val(InputBox("Please enter a value","Number program", ,50,50))
```



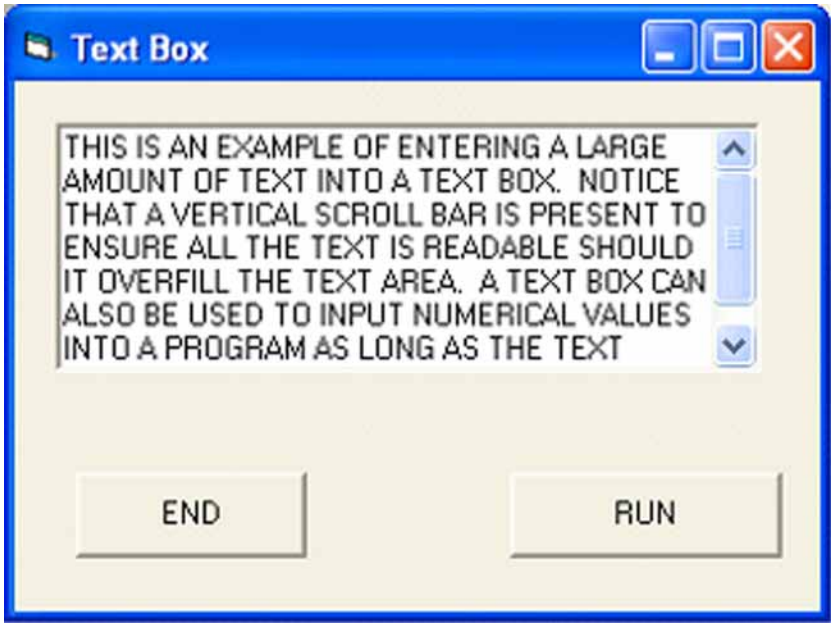
In this case the value `12345` is input to the program, a caption appears in the title bar and the InputBox appears at coordinates 50, 50

In most cases only the prompt is used.

Note that if a string contained `7902X7BMD` then the `Val` function would return the value 7902 and ignore the other characters.

2 Use of the Text Box

The main advantage of a text box is that a user can type in much more information during the execution of a program. Since text boxes can hold large amounts of text it is best that they are created using a vertical scroll bar.



By highlighting the text box and pressing function key F4 the properties window opens. From this window, the options `Vertical scroll bar` and `Multiline = True` are set.

If numeric values have to be input then, like InputBox, the `Val` function is used:

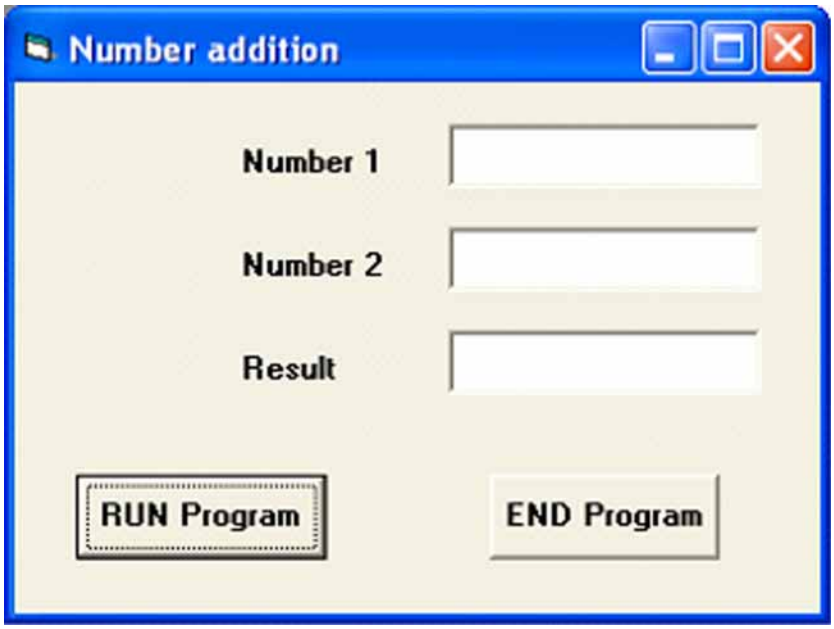
```
Number2 = Val(Text1.text)
```

Exercise 1

Open a new Visual Basic project and experiment with adding buttons and text boxes to the form from the toolbox window. Use the properties window to try-out various settings and captions for the buttons and settings for the text box and form.

Exercise 2

Open a new Visual Basic project and produce the following form that shows text boxes, labels and buttons. Use the properties window to create captions for the objects (labels and buttons are in bold).



Output methods

1 The MsgBox

The `MsgBox` function is useful in situations where the user requires, for example confirmation of an action. In its simplest form the format of the function is:

```
MsgBox  "The value entered was within the required range"
```

The output screen would look like:

The full format of the `MsgBox` statement is:

```
MsgBox "Prompt", Buttons, "Title"
```

`Prompt`: is the message to be displayed in the MsgBox

`Buttons`: is a numeric expression that specifies which buttons to display, with or without icons. The values are 0 (default), 1, 2, 3, 4, 5, 16, 32, 48 and 64.

`Title`: The string message displayed in the title bar.

Exercise 3

Experiment with MsgBox button values and create your own messages.

2 The Print command

The print statement is probably the most used output command since it can be used on its own or in combination with other objects and functions.

Used on its own the print statement with no explicit destination will produce output to the current window which, in most cases, is the Visual Basic form.

For example, if X = 5 and Y = 7 the following statement will output a text string and value to the form:

```
Print "The sum of the two variables X and Y is "; X + Y
```

If the form is populated with many objects it may be difficult to see the above output.

It is much better, therefore if the output is directed to a specified window such as a MessageBox or PictureBox. Although the latter object is meant for graphics it is also ideal for the output of lists of text. Using a PictureBox the previous print statement would become:

```
PictureBox.Print "The sum of the two variables X and Y is "; X + Y
```

Usually the term PictureBox is changed to PicBox, PicOutput, PicDisplay, Picture1 or whatever name is suitable for the output.

Exercise 4

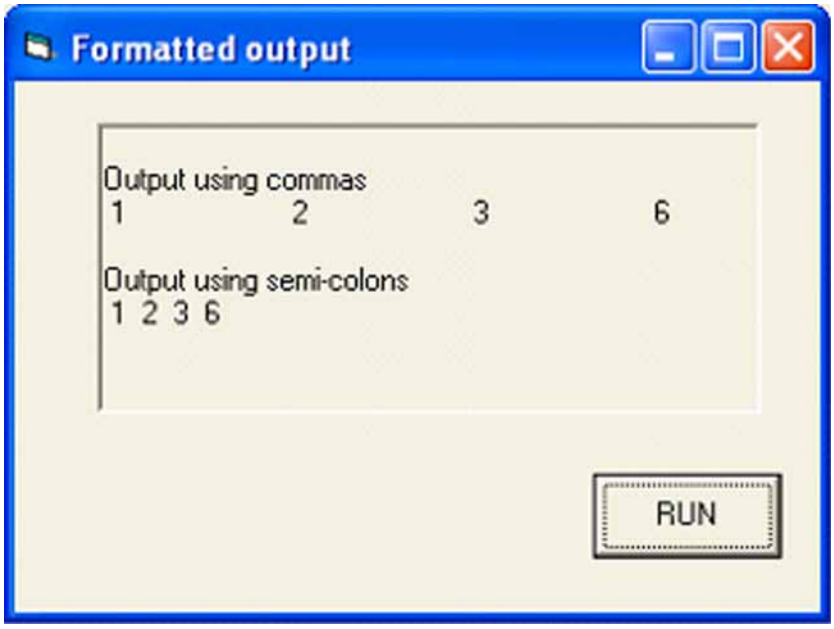
Create a new project and place a command button and PictureBox on the form. Rename the PictureBox as PicDisplay (F4 then change property 'Name').

Double-click on the command button and enter the following code. The first and last lines are present by default so they can be ignored when entering the text.

```
Private Sub Command1_Click()
  Dim number1 As Integer, number2 As Integer, number3 As Integer
  Dim Sum As Integer
  number1 = Val(InputBox("Please enter a value"))
  number2 = Val(InputBox("Please enter a value"))
  number3 = Val(InputBox("Please enter a value"))
  Sum = number1 + number2 + number3
  PicDisplay.Print
  PicDisplay.Print "Output using commas"
  PicDisplay.Print number1, number2, number3, Sum
  PicDisplay.Print
  PicDisplay.Print"Output using semi-colons"
```

```
    PicDisplay.Print number1; number2; number3; Sum
    PicDisplay.Print
End Sub
```

Run the program using your own values of number1, number2 and number3. Your output should look like the following:



Find out what the output would be if the Val function is removed from the `InputBox` statement.

Use of TAB() and SPC()

Two valuable print functions are TAB() and SPC() that allow for more formal results.

`TAB(6)` will begin output '6' units from the left margin

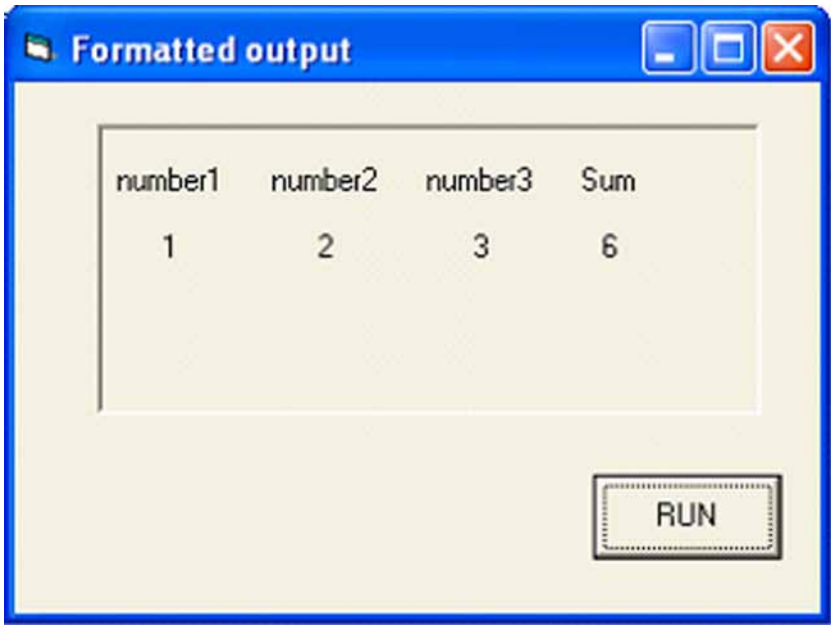
`SPC(6)` will output '6' units from the previous output.

Exercise 5

Modify the previous code as follows and run the program:

```
PicDisplay.Print
PicDisplay.Print Tab(2); "number1"; Tab(14); "number2"; Tab(26); "number3"; Tab(38);
    "Sum"
PicDisplay.Print
PicDisplay.Print Spc(4); number1; Spc(9); number2; Spc(9); number3; Spc(7); Sum
```

This time you end up with the output that should look like the following:



You will learn more about formatting values later.

6.4 Program Structure

A typical Visual Basic program takes the form shown in Code 6.1

```
Option explicit
'General area
'Public or Global variables declared here
-----------------------------------------------------

Private Sub cmdClick_Click()
constant definitions
  'the constants used in the program

variable definitions
   'the variables used in the program

   'main parts of the program

INPUT PHASE

   'read in data from keyboard, file, , mouse etc

CALCULATION

   'process the data to give us the results we want
```

```
OUTPUT
   'send results to a window, printer, file, disk etc.

End sub
-------------------------------------------------

Sub procedures()
'Procedure and function definitions
End Sub
```

Code 6.1

The program area contains the following:

A *general area*: this area is public and variables declared here will be 'seen' by all parts of the program. More on this will be discussed later under the term *scope*.

A *sub procedure cmd_click( ):* this contains the main program code that will be initiated in the event of a mouse click, for example. Within this code there are further declarations local to the sub procedure. The declarations are:

Constants: any fixed values which will stay *constant* throughout the program are declared here;

Variables: variables other than global variables that are going to be used in the program are declared here.

Sub procedures: any subprocedures required by the program are declared after the main sub procedure. These are described more fully later on.

Functions: - any functions the program will use are also declared here. These are described more fully later on.

The input may be from a variety of devices, not just the keyboard. A disk is an input device when the computer reads data (including programs) from it. The program may also receive data internally during program execution.

The calculation is the heart of the program. The input data is transformed to the output data. This, in short, is all that a computer does.

The output phase is less straightforward! There is no screen output in Visual Basic, only windows called forms. Output can also be directed to a file, printer or disk.

You will see example output methods in the program exemplars.

6.4.1 Example program

Suppose, for example that you were going to write a program to calculate the area of a rectangle, where the user will input the dimensions and the program will return the area. The program will take the general form as shown in Code 6.2.

```
Option Explicit
------------------------------------------------

Private Sub cmdDisplay_Click()

  Dim Length As Integer
  Dim Width As Integer
  Dim Area As Integer

  Length = Val(Lngth.Text) 'Input length
  Width = Val(Wdth.Text)   'Input width
  Area = Length * Width    'Calculate area
  Ar.Text = Str$(Area)     'Output result

End Sub
```

Code 6.2

This program uses text boxes to input and output the data.

1. In the program the variables *Length, Width* and *Area* are declared using the `Dim` statement (see later)
2. Values of Length and Width are input via text boxes. The Visual Basic function *Val* changes text into numeric values.
3. The area is calculated
4. Result output to a text box. The function Str$ converts the numerical value back to a string value to be recognised by the text box.

Figure 6.3 shows a program run:

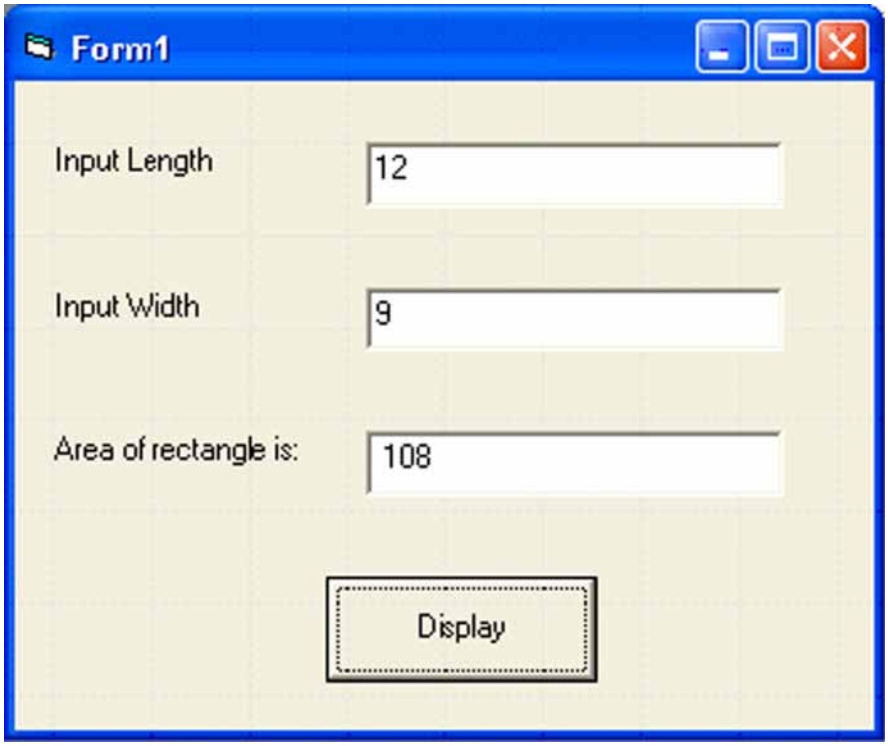


Figure 6.3:

You can see that the form contains 3 labels and 3 text boxes for the input and output of data and a control button.

6.5 Data types

Visual basic has a number of in-built data types, the significant ones for this topic being integer, single (real), boolean and string.

These are summarised with others in Table 6.2

Table 6.2:

Data Type	Optional Suffix	Description
`Integer`	%	These are whole numbers, positive or negative, without a decimal point. The range depends on the machine but if 16 bits long the range is -32,768 to +32,767.
`Long (integer)`	&	Extended range of integer from -2,147,483,648 to +2,147,483,647
`Single (real)`	!	These are floating point numbers, e.g. 3.33333. When 32 bits long their range is $\pm$3.4e $\pm$38, and referred to as *single-precision*.
`Double (real)`	#	*Double-precision* floating point numbers within the range $\pm$1.7e $\pm$308.
`String`	$	Fixed length up to 65,400 characters Variable length up to 2 billion characters.
`boolean`		This data type can represent two values: *true* and *false*.
`variant`		Any numeric value up to the range of a double-precision number or any character text.

Note that:

1. Although the range of floating point numbers is huge, they are not all that accurate - often not much more than 8 - 16 digits (depends on the machine and compiler). The rest of the number you may see on the screen is an approximation. For accuracy you need to use integers.
2. The data type 'variant' is the default mode in Visual Basic and will be discussed later in the topic.

Identifying data types

On the Web is a interactivity. You should now complete this task.

6.6 Visual Basic Nomenclature

An important aspect of any language is the rules for naming the objects such as *constants, variables, subprocedures* and *functions*.

There are *five* rules in Visual Basic which prescribe *valid* variable names. All Visual Basic variable names:

1. must always begin with a letter
2. must contain no spaces
3. must only consist of letters or digits with no embedded periods or other punctuation

marks except the underscore character

4. can be no longer than 255 characters
5. cannot use Visual Basic reserved words (see Table 6.5).

The variable names shown in Table 6.3 are valid:

Table 6.3:

temp1	velocity	unit1
pay_period	fred	day_month_year

The variable names shown in Table 6.4 are not valid. Why?

Table 6.4:

40th_birthday	@discount	%_rate
constant	bad-variable	string

In addition, a number of **keywords** are reserved in Visual Basic for use as commands. Some of these keywords are listed in Table 6.5 and ***must not*** be used as variable names.

Table 6.5:

`dim`	`else`	`true`	`false`	`to`
`sub`	`function`	`while`	`wend`	`private`
`const`	`integer`	`for`	`string`	`elseif`
`case`	`public`	`repeat`	`program`	`and`
`procedure`	`boolean`	`until`	`clear`	`or`
`if`	`string`	`do`	`print`	`not`

A more complete list of keywords can be found in any good Visual Basic manual.

Sentence completion - variables

On the Web is a interactivity. You should now complete this task.

6.6.1 Review Questions

Q6: Which one of the following variable names would not be allowed in Visual Basic?

a) Hello
b) Number_1
c) 17.5%Vat
d) SCHOLAR

Q7: The choice of a variable name in programming is very important because (choose one):

a) It is easier for programmers to locate and fix errors
b) It makes the programs more efficient
c) Meaningful variable names make the program more reliable
d) All of the above

Q8: A program is written to input a person's age and calculate how old they will be in 10 years time. Which of the following data types would be required in the program?

a) String
b) Real
c) Boolean
d) Integer

Q9: How many variables would the program require?

a) 1 or 2
b) 3 or 4
c) 4 or 5
d) greater than 5

Q10: A program is designed to generate prime numbers up to the maximum value possible with precision. The results are stored in a variable called *prime*. In order to perform the calculation the variable prime would need to be declared as:

a) Integer
b) Real (double)
c) Integer (long)
d) Real(single)

6.7 Declaring Variables

Within Visual Basic variables are used to represent and identify values within a program. The values are held in temporary storage locations in the computers memory.

Each memory location is identified by a unique variable name, and the value of its contents can change during the execution of a program.

Once the program has ended variables will be reset; all numeric values become zero and strings become empty.

For this reason it is important that all variables are assigned initial values before a program run.

To use a variable in Visual Basic, three quantities must be specified:

1. Name of the variable
2. Type of variable
3. Value of the variable

Variables are declared using the `Dim` statement which allocates temporary storage to them. It is usual to use `Dim` statements before any other code so that total memory can be allocated at run time.

Examples of Dim statements are:

Dim *bin*Found As boolean
Dim *int*Maximum As integer
Dim *dbl*Precision As double
Dim *str*Myname As string

Notice that each variable name has a prefix attached. This is a Visual Basic construct that allows the programmer to differentiate between the various types of data objects. Although not essential It is good programming practice to do so and also to use *meaningful variable names* so that values can be easily identified in programs.

Alternatively the initial letters of the prefix can be used instead as long as readability is not compromised.

The main data type prefixes are summarised in Table 6.6:

Table 6.6:

Data type	**Prefix**
Boolean	bin or b
Integer	int or i
Long	lng or l
String	str
Single	sng
Double	dbl or d
Constant	con or c

Exercise

Which of the following variable assignments match?

a) intValue = 89
b) sNumber1 = 3.1417
c) strName = "Hello"
d) iNumber2 = 25.683
e) strString = 65.310
f) lonBig_Number = 5.6432e10
g) bFound = false

6.7.1 Implicit and Explicit declaration

Visual Basic offers two levels of variable type declarations:

1. Implicit declaration
2. Explicit declaration

Implicit declaration

In **implicit declaration**, if the 'dim' statement is used on its own without assigning type to a variable or a variable is assigned a value, then Visual Basic will assign the data type variant. This is the default setting that Visual Basic will assign variables if not declared as some other type.

Visual Basic does not force you to declare data types but it is much better if you do.

For example consider the following lines of code as shown:

```
Private Sub Form_Load()
Dim MyText
MyText = "Sample program"
FirstNumber = 5
SecondNumber = 10
Total = FirstNumber + SecondNumber
End Sub
```

FirstNumber, SecondNumber and *Total* have not been declared but Visual Basic has assigned them data types 'on the fly' the first time they encountered. Although *MyText* has been declared, it has no type. All variables have therefore been assigned the type *variant*.

But beware! If a *variant* variable is mis-spelled later in the program then a new variable will be created by Visual Basic. This is a common error and can create program bugs that are difficult to find.

Explicit declaration

In **explicit declaration** each variable is declared unambiguously using the `Dim` statement.

It is recommended that this option be used at all times. This helps to prevent errors and allows the computer to work more efficiently. If Visual Basic knows the data type through declarations then the requisite amount of memory can be assigned thus making memory management more effective. Programs with defined variables also run around 3 times faster than those with variant data types.

Visual Basic can be forced to make variable declaration the default setting. By clicking on *Tools* then *Options* a window like Figure 6.4 will open. Checking the *Require Variable Declaration* box will ensure Visual Basic starts up in explicit declaration mode.

Note: Visual basic.NET does not support the type variant.

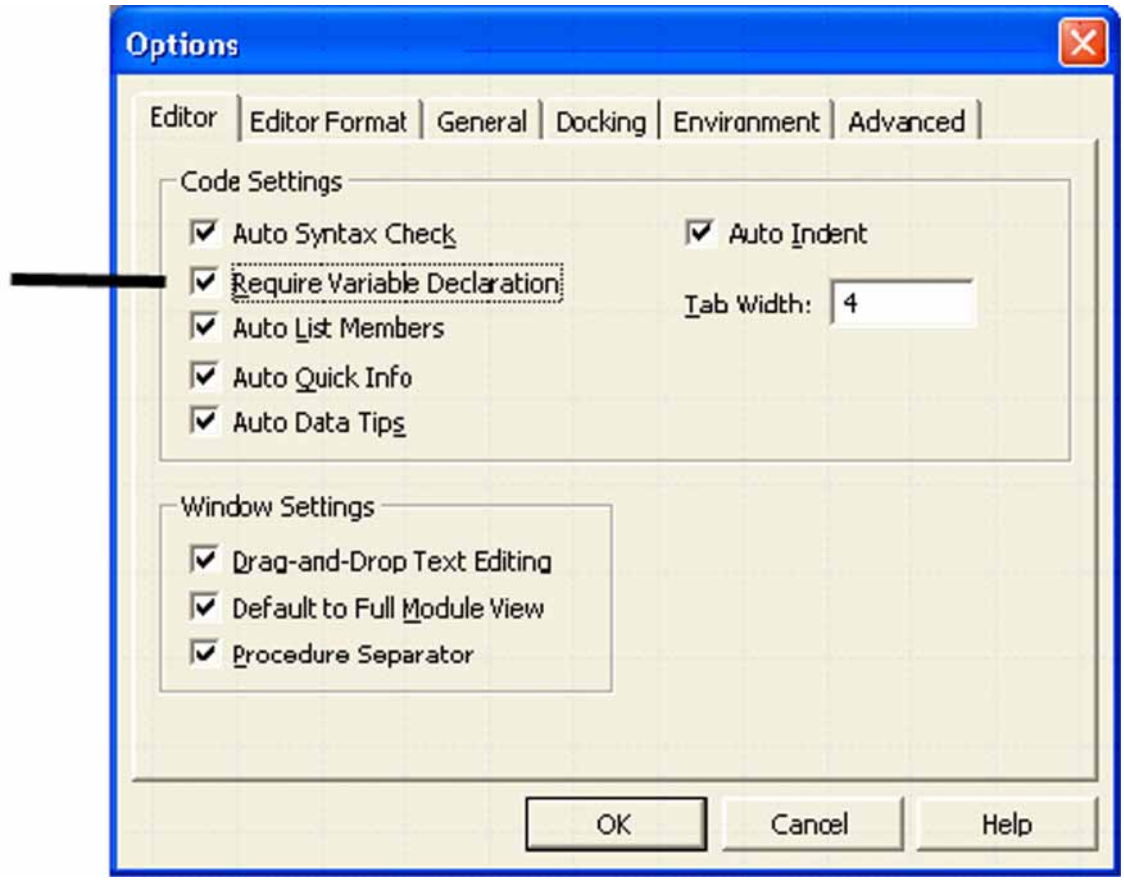


Figure 6.4:

The program of Code 6.2 can now be re-written as:

```
Option Explicit

Private Sub Form_load()

Dim FirstNumber As integer
Dim SecondNumber As integer
Dim Total As integer
Dim MyText As string

FirstNumber = 5
SecondNumber = 10
Total = FirstNumber + SecondNumber

MyText = "Sample Program"

End Sub
```

Code 6.3

6.8 Declaring constants

A constant is a quantity that is allocated by the user, usually at the start of a program with variables, although they can be defined anywhere in the program. A constant retains the same value throughout program execution and cannot be altered.

Constants are declared using the const statement to create string or numeric values

Examples of constant declarations are:

```
Const cPi As single = 3.141592

Const cAvogadro = 6.023E23

Const cName = "This is my name"
```

Like variables, if a constant is not defined then it will be assigned the *variant* type.

Some programmers like to express constants all in capital letters to differentiate them from variables. This is an individual preference. This makes no difference to the program whatsoever; its only purpose is to bring to your attention the fact that this particular name is being used for a constant and not a variable.

6.8.1 Example program 1 - Calculating the circumference of a circle

Problem: Write a program that calculates the circumference of a circle, when supplied with the radius. The value of PI is defined as a constant.

Solution: A typical solution to the problem is shown in Code 6.4. The circumference is calculated using the formula: $2 \times \text{PI} \times \text{r}$.

Note. It is good programming practice to include comments in every program. The information allows better readability of the program and also breaks the code listing into meaningful chunks.

The comment symbol is the single quote ('). Visual Basic ignores all text that comes after the quote. Comments can also be embedded within the lines of code.

```
Option Explicit
Private Sub Command1_Click()

'program circle

'10th February 2004
'Program by Fred Fink

'This program prompts the user to enter a value for the radius
'of a circle.  It then uses the formula: 2*PI*radius to
'calculate the circumference

Const PI As Single = 3.14159      'Declare constant PI and assigns value

Dim radius As Single
```

```
Dim circumference As Single

radius = InputBox("Please enter the radius of the circle")

circumference = 2 * PI * radius

Print "The circumference of circle with radius :";
radius; " is: "; circumference

End Sub
```

This file (Circle.txt), can be downloaded from the course web site.

Code 6.4

In this program the input and output commands are controlled using the *InputBox* function.

The program output Figure 6.5 and Figure 6.6 are shown below:

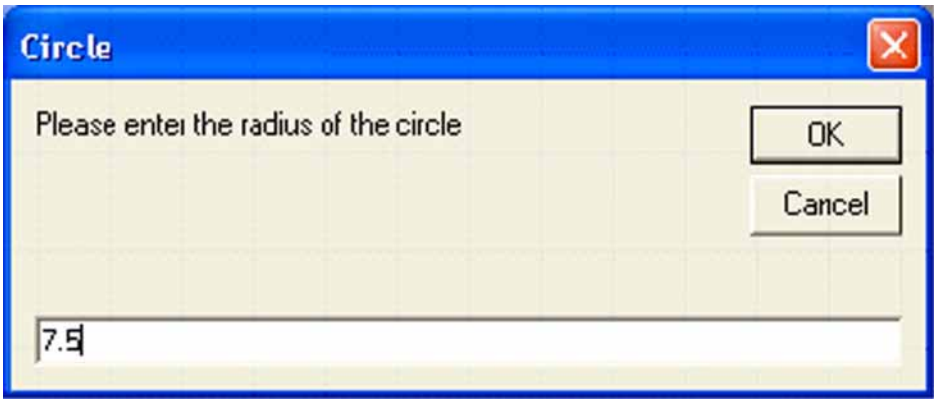


Figure 6.5:

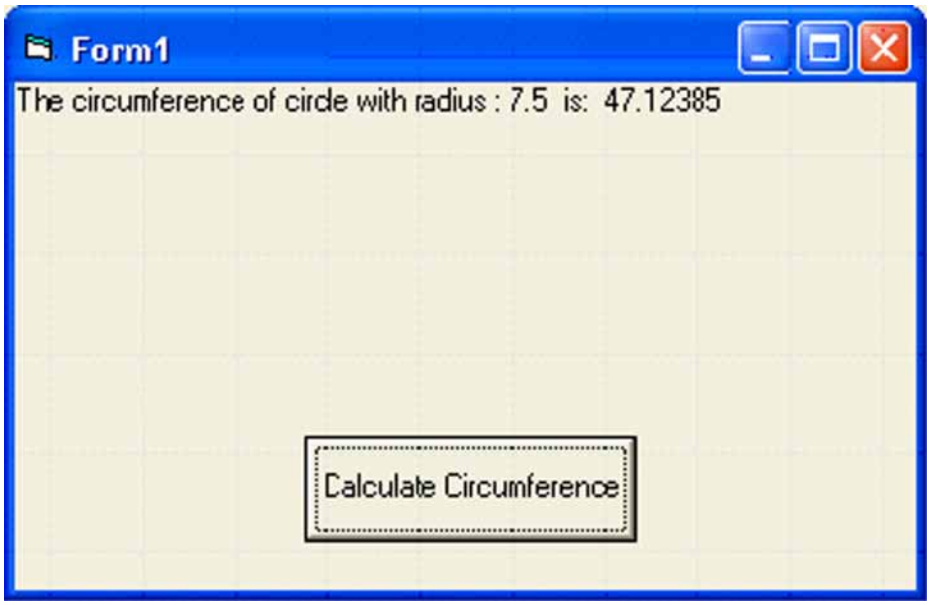


Figure 6.6:

Note that this method uses multiple windows. Input is via an inputBox and output is on the form itself, using the *print* command.

Compare this to the method used in the *rectangle* program.

6.8.2 Example program 2 - Use of a boolean variable

Problem: A program is written to display whether a number entered at the keyboard is greater than or equal to zero. The program will prompt the user to enter a number, read the number entered at the keyboard. It will then display a message, together with the value true or false, depending on the value of the number entered.

Solution:

The algorithm is shown below:

```
1. issue message to enter number
2. read the number typed in at the keyboard
3. set the boolean variable to true or false
   depending upon number >= or number < 0
```

The full Visual Basic code for this program is shown in Code 6.5:

```
Private Sub cmdRun_Click()
'10th February 2004
'Program by Fred Fink

Dim number As Integer
Dim result As Boolean


' This is a program that will prompt the user for a number
' It will then check whether the number is less than or equal
' to zero or greater than 0
' A boolean variable will hold the result.  A message saying
' true or false will be printed on the screen to say whether it is
' greater than zero

'Start of main program

number = InputBox("Please enter a number")
result = number >= 0
Print number; "greater than or equal to zero = "; result

End Sub
```

Code 6.5

In this program the InputBox function is used to input the data, as before. The output is shown in Figure 6.7:

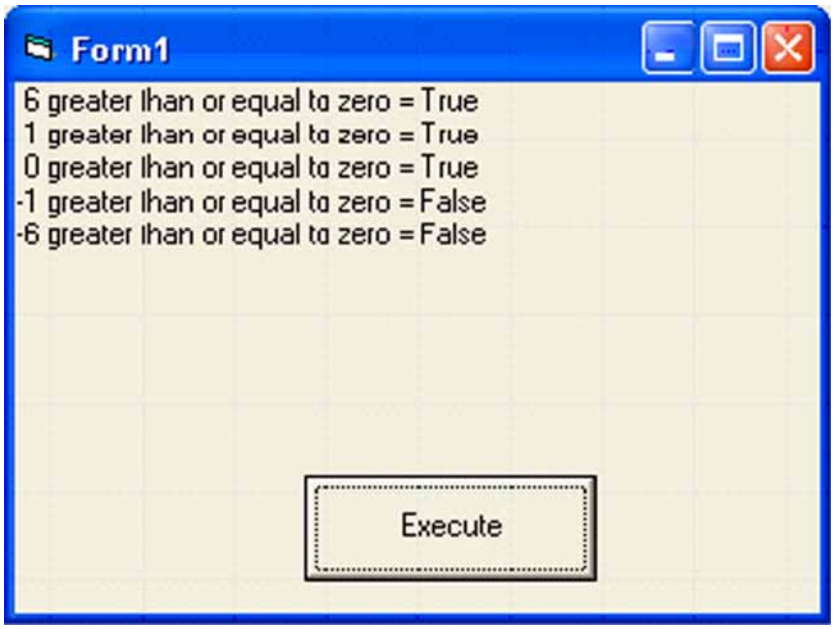


Figure 6.7:

Boolean values are easy to use, and when used, they can often make a program more readable. They are often tested to control the flow of execution of a program.

When defining a list of variables with the same type specifier, the variables can all be defined on the same line as the type specifier or separate type specifiers can be used for each variable. For example:

```
Dim number1 As integer, number2 As integer, sum As integer
```

is equivalent to

```
Dim number1 As integer
Dim number2 As integer
Dim sum As integer
```

However the statement:

```
Dim number1, number2, sum As integer
```

will assign *sum* as integer with *number1* and *number2* being assigned *variant*.

6.8.3 String variables and functions

A *string* is simply a collection of characters that are defined within quotes. The following are examples of string values:

"coffee"

"This program is not working"

"xyz"

"February 16,2004"

"Visual Basic 6"

"Shrove Tuesday is pancake day"

"51"

The quotation marks are very important since they tell Visual Basic that the enclosed characters are a *string variables*. Characters not enclosed in quotation marks are considered to be a numeric variable or some other part of the Visual Basic language.

Note. The string value "51" does not equal the value 51. You might think that it's a value for an integer, but it's not. However there is a Visual Basic function, VAL, that will convert string variables to values. The following statement:

```
NumericalValue  = VAL("51")
```

will allow the variable `NumericalValue` to be assigned the integer value `51`.

Of course the reverse is also true. Given a value, it can be converted into a string using the Visual Basic `STR$` function. The following statement:

```
StringValue  =  STR$(51)
```

will allow the variable `StringValue` to be assigned the string `"51"`

Two other useful functions are `CHR$` and `ASC`. The following examples show how they are used:

```
LetterValue  =  ASC("A")
```

will assign the value 65 to `LetterValue`. This is the ASCII code for the letter "A".

```
Letter$ = CHR$(66)
```

will assign the string "B" to `Letter$` since the ASCII code for "B" is 66.

Table 6.7 summarises the string functions:

Table 6.7:

Function	Usage	Examples
`VAL`	Converts string to value	`a = Val("66")`
`STR$`	Converts value to string	`b$ = Str$(77)`
`ASC`	Converts string to ASCII value	`c = ASC("D")`
`CHR$`	Converts ASCII value to string	`d$ = CHR$(68)`

What are the results of each example?

6.8.4 Concatenation

Concatenation is simply joining string variables together to make longer strings.

The operator is the ampersand symbol (&). When two or more strings are combined the second strings are added directly to the end of the preceding string. The result is a longer string containing the full contents of both source strings.

The following example shows concatenation structure:

```
NewString = StringOne & StringTwo & StringThree
```

Here `NewString` represents the variable that contains the result of the concatenation

operation. `StringOne`, `StringTwo`, and `StringThree` all represent string variables.

Note that the ampersand must be preceded and followed by a space.

Example of concatenation

Example 1

The statement:

```
Print "Man" & "chester"
```

would produce the string `"Manchester"`

Example 2

```
String1  = "This is "
String2  = "con"
String3  = "cat"
String4  = "en"
String5  = "nation"
```

`Print String1 & String2 & String3 & String4 & String5` will produce as output:

`"This is concatenation"`

Practice in using simple variables

Now let us put this to some practical use by having a look at an example program.

Example : Adding values

Problem:

Describe the operation and the output of the following program.

```
Dim number1 As Integer
Dim number2 As Integer
Dim sum As Integer


number1 = 0         {initialise the variables to some value }
number2 = 0
number1 = InputBox("Give me a number")
number2 = InputBox("Give me a second number")
Print
sum = number1 + number2
Print("The sum of :"; number1;" and "; number2; "is: ";sum)
```

This file (add.txt), can be downloaded from the course web site.

Code 6.6

Solution:

The program defines 3 integer variables. Values are assigned to two of them. These two variables are then added together with the result being stored in the third variable. The result of the sum is then displayed.

Given number1 = 32 and number2 = 27, the output of the program is:

```
The sum of : 32 and 27 is : 59
```

In this program the variables `value1`, `value2` and `sum` are declared to be of data type integer. This declaration statement must occur before the variables are used in the program since it instructs the compiler to allocate sufficient memory storage for each of the data items on the list. This is important. You cannot use a variable unless you have declared it first. Also remember that variables may *not* be initialised when they are declared. They may not be set to zero, but have random numbers in them. In the case of this program you either gave the variables specific values, by reading values from the keyboard or you gave `sum` a value which was calculated by the program. In either case the variable had been given a value before it was used.

Q11: What is wrong with this Visual Basic program?

```
'program subtract

Dim number1 As Integer
Dim result As integer

number1 = InputBox("Please enter the first number")
number2 = InputBox("please enter the second number")
result = number1 - number2
Print("Number 1 take away number 2 = ";sum)
```

20 min

Calculating minutes

Learning Objective

Be able to use get input from the user.

Be able to use arithmetic operators.

Be able to display output on the screen.

Write a program which prompts the user to enter a number of days, hours and minutes. The program will then calculate and display this as a total number of minutes.

6.9 Variables and scope

Variables have another important characteristic called scope. This determines which parts of a program are able to 'see' the variable and change its value.

The scope of a variable is determined not only by the type of declaration but also the declaration's location. For instance, the Dim keyword assumes different meanings in different parts of a form's code.

In Visual Basic variables can be declared as shown in Figure 6.8:

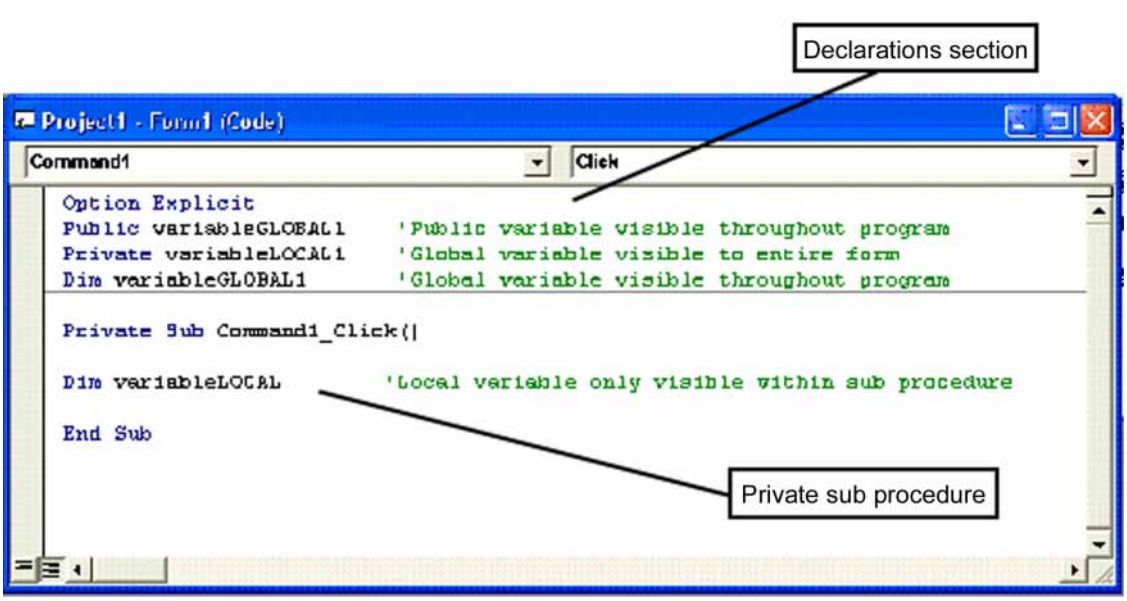


Figure 6.8:

1. If a variable is declared in the general declarations section with the `Dim` statement then it is Public or Global to the entire program.
2. If it is declared as *Public* in the general declarations section then it is *Global* to the entire program.
3. If it is declared *Private* in the general declarations section then it is *Global* to the entire form and modules.
4. If declared within the sub procedure then the variable becomes Local to that sub procedure and will not be recognised elsewhere.

Consider the following program in Code 6.7:

```
Option Explicit
'Program to demonstrate global and local variables

Public TestVariable As Integer
Dim TestString As String

Private Sub ExecuteMain_Click()
  TestVariable = 55
```

```
  TestString = "Hello 1"
  Print "In main program Testvariable = "; TestVariable
  Print "In main program TestString = TestString"
End Sub


Private Sub ExecuteLocal_Click()
  Dim TestVariable As Integer
  Dim TestString As String
  Print
  Print "Call 1 gives TestVariable = "; TestVariable
  Print "Call 1 gives TestString = "; TestString
  Print
  TestVariable = 110
  TestString = "Hello 2"
  Print "Call 2 gives TestVariable = "; TestVariable
  Print "Call 2 gives TestString = "; TestString
End Sub
```

This file (Scope.txt), can be downloaded from the course web site.

Code 6.7

`TestVariable` and `TestString` are declared `Public`.

They are given values in sub procedure `ExecuteMain_Click()`

They are re-declared in sub procedure `ExecuteLocal_Click()` and assigned new values.

Figure 6.9 shows the program output. The program is called twice by activating the two buttons.

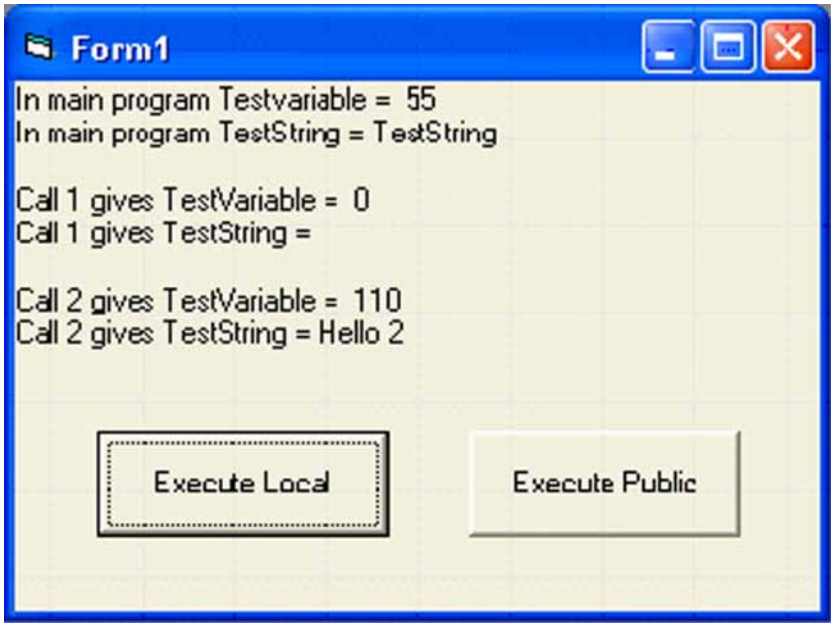


Figure 6.9:

Public gives the original values(that can change).

In call 1 local procedure resets the variables to 0 (since they have been re-declared).

Call 2 produces the new values from the local procedure.

Therefore, local variables always win!

Note: Extensive use of global variables is bad practice; the use of global variables should be kept to a minimum since their values can change from anywhere within a program as you have just seen.

Local variables on the other hand offer the following advantages:

1. side-effects caused by a subprogram altering the value of a variable used elsewhere in the program are greatly reduced
2. debugging is easier since access to variables is localised and so tracking errors can be faster
3. the transfer of procedures and functions from one program to another is simplified.

Sentence completion - global and local variables

On the Web is a interactivity. You should now complete this task.

6.9.1 Review Questions

Q12: In high level language code a variable declared outside a procedure is (choose one):

a) Local
b) Constant
c) Global
d) Functional

Q13: Which one of the following describes the relationship between main memory and program variables?

a) Name of variable identifies memory location
b) Data type defines how much memory is needed
c) Values of variables are held in main memory
d) All three statements above

Q14: Which one of the following describes a **local** variable?

a) Can be accessed anywhere in the program
b) They are hidden from other procedures and functions
c) Their values can easily be altered by mistake
d) The use of local variables is frowned upon

Q15: The scope of a variable is an important aspect in programming. Scope is best described as:

a) The range of values the variable can cope with
b) The amount of memory required by the variable
c) The extent to which the variable can be 'seen' by the rest of the program
d) The number of times the variable can be used in the program

Q16: Which one of the following describes a boolean variable?

a) It holds a numerical value
b) It cannot be used in expressions
c) It is only used in looping structures
d) It can have only the values true or false

6.10 Operators

Assignment operator

The assignment operator in Visual basic is equals (=) sign. The general form for the assignment operator is:

```
variable = expression
```

This statement says make left hand side equal to the right hand side. It makes the variable take on the same value as expression. You have already seen this used in previous programs.

For example:

```
circumference = 2 * PI * r
```

where the result of the expression `2* PI * r` is assigned to the variable `circumference`

Arithmetic operators

Table 6.8 list the main arithmetic operators in Visual basic:

Table 6.8:

Arithmetic operators	**Example**	**Result**
+ (add)	16 + 14	30
- (subtract)	27 - 9	18
/ (real division)	27.83 / 3	9.27
* (multiply)	12.65 * 5	63.23
\ (integer division)	16 \ 7	2
mod (modulus operator)	25 mod 7	4
^ (raise to the power)	10 ^ 3	1000

It is important to remember that integer division gives you a whole-number answer only - any remainder is discarded. For example:

5 \ 2 gives 2 as the answer;

16 \ 5 gives 3 as the answer.

The modulus operator is used for division to obtain the remainder. The expression

```
x mod y
```

produces the remainder when `x` is divided by `y`

The `mod` operator can be applied to integers and also with `real`. For example:

5 mod 2 gives 1

8 mod 3 gives 2.

9 mod 3 gives 0 (zero).

7.1 mod 3.1 gives 1 (numbers rounded down)

10.8 mod 3.6 gives 3 (numbers rounded up)

Make sure you understand why.

6.10.1 Operator precedence

When several operations occur in an expression, each part is evaluated and resolved in a predetermined order called operator precedence. Parentheses can be used to override the order of precedence and force some parts of an expression to be evaluated before other parts. Operations within parentheses are always performed before those outside. Within parentheses, however, normal operator precedence is maintained.

When multiplication and division occur together in an expression, each operation is evaluated as it occurs from left to right. Likewise, when addition and subtraction occur together in an expression, each operation is evaluated in order of appearance from left to right.

Arithmetic operators are evaluated in the following order of precedence:

Arithmetic operator precedence	
Parentheses ( )	1
Exponentiation (^)	2
Negation (-)	3
Multiplication and division (*, /)	4
Integer division (\)	5
Modulus arithmetic (Mod)	6
Addition and subtraction (+, -)	7

Example 1 - Operator precedence: 4 + 5 * 2

Problem: What is the result of 4 + 5 * 2?

Solution: The multiplication is calculated first, then the addition since multiplication has a higher precedence than addition, e.g.

```
  4 + 5 * 2
= 4 + 10
= 14
```

and so 4 + 5 * 2 is equal to 14.

Example 2 - Operator precedence: (4 + 5) * 2

Problem: What is the result of (4 + 5) * 2?

Solution: The expression in the brackets is calculated first, then the multiplication - the brackets have altered the order of precedence, e.g.

```
  (4 + 5) * 2
= (9) * 2
= 18
```

and so (4 + 5) * 2 is equal to 18.

Example 3 - Operator precedence: 3 + (7 - 5)^2 * 4

Problem: What is the result of 3 + (7 - 5)^2 * 4?

Solution: The expression in brackets is evaluated first, then exponentiation, followed by multiplication then addition, i.e

```
  3 + (7 - 5)^2 * 4
= 3 + 2^2 * 4
= 3 + 4 * 4
= 3 + 16
= 19
```

and so 3 + (7 - 5)^2 * 4 is equal to 19

Matching operations and results

On the Web is a interactivity. You should now complete this task.

6.11 Programming constructs

Structured programming is based on three constructs:

- sequence
- selection
- repetition

6.11.1 Sequence

This is the computer's basic mode of operation. This is how it's designed to work. A computer is designed to carry out an instruction and move automatically onto the next instruction.

A program is working in sequence when it

- begins at the beginning
- carries out each statement or instruction once and only once
- stops when it's reached the end

In other words, it misses nothing out (which happens under selection) and it does nothing more than once (which happens under repetition).

Sequence is the default mode of operation for a program: a program works in sequence unless it is explicitly told to do otherwise by means of one of the other programming constructs. Left to itself, it will always carry out a statement and proceed automatically to the next.

Example of simple sequence

The program examples you have seen so far follow simple sequences.

Try the following activity.

Problem: A program is written to ask the user to enter a number at the keyboard. The program then doubles it and displays the result on the screen. Code the following algorithm in Visual Basic and run the program:

```
1. request a number from the user
2. get the number typed in by the user
3. double the value of the number
4. display the new value on the screen
```

Sequential control of a program limits the type and range of problems that can be solved. Sequence is basic and essential, but limited. A program in simple sequence always does the same sort of thing, and it can't easily do things more than once.

We want programs that can carry out tests and make decisions. We want programs that can do things over and over again.

With the use of selection and repetition, the range of problems that can be solved becomes much larger. This is what we will look at next.

6.11.2 Selection

A fundamental task in any program is the decision of what to do next. Control constructs enable you to make decisions in your programs to determine when certain parts of the program will be executed.

In a similar manner to English you can make decisions using the if statement - for example, if it is raining then I will take my umbrella. These simple decisions can then be further extended as will be considered in this section.

In this topic you will look at the following control constructs and how to use them to enable decisions to be made in programs.

- the `if` statement
- use of logical operators in decision-making
- the `if...else` statement
- nested `if` statements
- multi-way branching with `elseif`
- the `case` statement

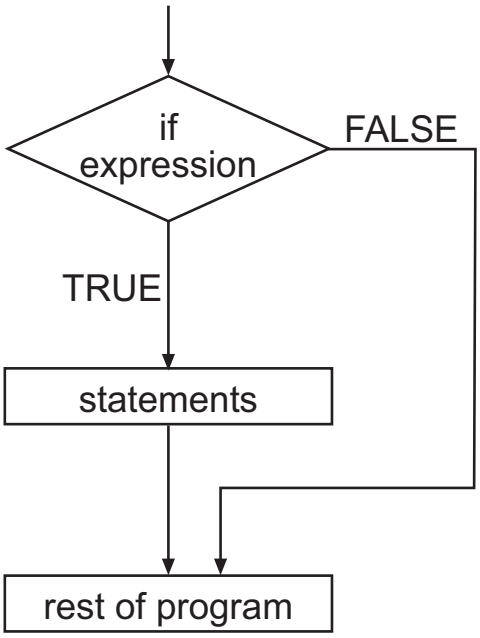
6.12 The IF Statement

The general decision making capability in Visual Basic is provided by the `if` statement. The format of this statement is:

```
if condition then statement
```

or

```
if condition then
statements
end if
```



The program statement may be a single statement or may be a block of statements.

When the expression is true, the statements following are executed. When the expression is false, then none of the statements are executed and control passes to the remainder of the program

Example 1 - Using a simple IF statement

Problem: A program is written to ask the user for a number. If the number entered by the user has the same value as a value stored in a constant, then the program will display a suitable response.

The algorithm is shown below:

Solution:

```
1. request a number from the user
2. get number typed in at the keyboard
3. IF (the number typed in  = the program constant) THEN
4.   display "I like 99s"
5. display "Program End! So you don't like 99s "
```

The full Visual Basic code for this program is as follows:

```
Option Explicit
Private Sub cmdExecute_Click()
  'program simpleIf_Then

  '13th February 2004
  'Program by Fred Fink

  'A program to show a simple if..then statement

  Const Ice As Integer = 99
```

```
  Dim userno As Integer

  userno = InputBox("Give me a number")

  If userno = Ice Then

    PicDisplay.Print "I like "; Ice; "s"

  End If

  PicDisplay.Print "Program End! "
End Sub
```

Code 6.8

In this program input is via an InputBox and output by means of the *PictureBox* function. Although a *PictureBox* is meant for graphics it can deal with text as well and this somewhat tidier than displaying text on a form.

The *PictureBox* property is set to the name *PicDisplay* in the properties window.

Program output is seen in Figure 6.10:

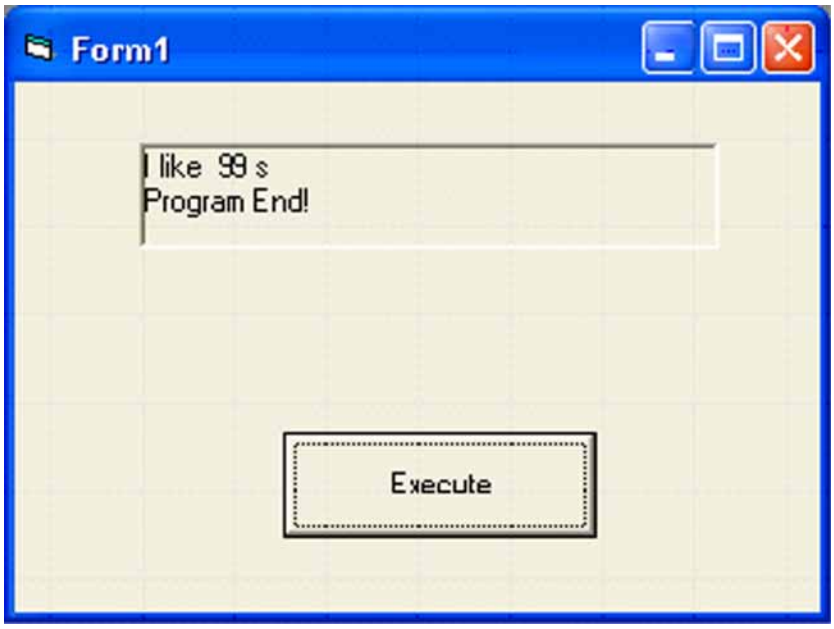


Figure 6.10:

Several examples are given here that use comparison operators and `if` constructs within a program to solve a problem. Try writing the programs for yourself and experiment with making changes to the code to ensure you understand what is happening.

Example 1 - Calculating the area and circumference of a circle

Problem: Write a program that calculates the area and circumference of a circle. The calculations should only be performed if the radius is positive (i.e. a valid radius).

Solution: In the previous topic an example was given for calculating the circumference of a circle using a constant for the value of pi. This example could be modified to use an

if statement to validate the radius i.e

```
if radius > 0 then
  do calculations
display results
```

Example 2 - Write a program to input numbers from the keyboard and to print them out. The program terminates when -1 is entered. A suitable message should be output.

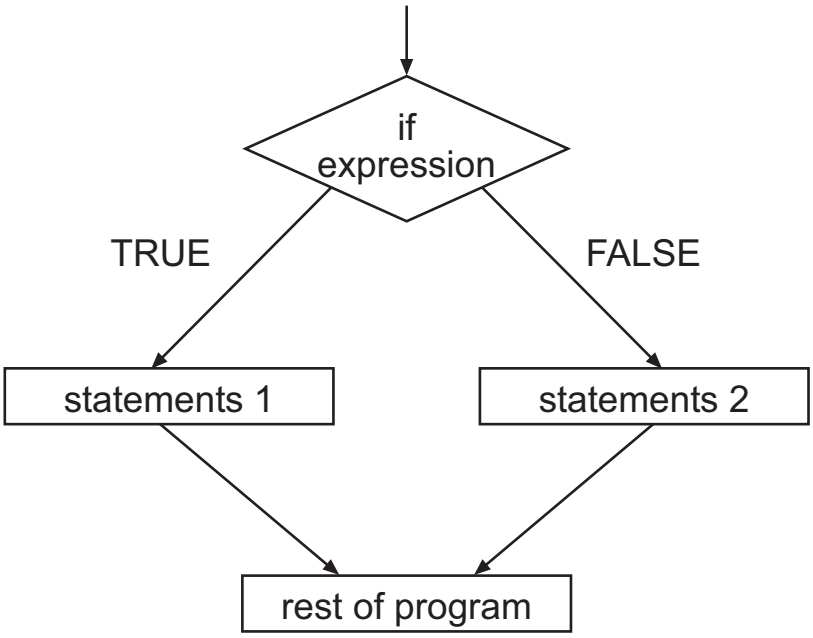
```
input number
if number > 0
     output number
end
terminate program
output message
```

6.13 The If.. Then.. Else Statement

So far we have shown you the use of the if..then statement when there is only one alternative i.e. if the expression is true then execute a single code fragment. If the expression is not true, you may want to execute a different code fragment. This is known as an if statement with two alternatives and the general format becomes:

```
if condition then
    statements 1
else
    statements 2
End if
```

Again the statements may be single or they can be multiple statements.



If the expression is true, then the code fragment following the then (statements 1) will be executed. If the expression is false, the statements following the else (statements 2) will be executed.

Example - Single or double?

Problem: A program is written to prompt the user for a number. If the number entered by the user has a single digit, then the program will display an appropriate message telling the user that this is single-digit number. Otherwise it will output the message "double-digit number"

The algorithm is shown below:

Solution:

```
1.Ask for user to input number between 1 and 99
2.if the number < 10
3.display "single digit number"
4.else
5.display "double digit number"
```

The full Visual Basic program is seen in Code 6.9

```
Option Explicit
Private Sub Command1_Click()

'program simple_If_Then_Else

'15th February 2004
'Program by Fred Fink

'This program will prompt the user to enter a number
'The program will test the number to see if it is
```

```
'less than 10 and print the message that it is a single-digit number.
'Otherwise it will output "double-integer number"

Dim testNumber As Integer

testNumber = InputBox("Please input a number ")

If testNumber < 10 Then
   PicDisplay.Print testNumber; "is a single digit number"
       Else
    PicDisplay.Print testNumber; "Is this double-digit number"
End If

End Sub

Private Sub EndProgram_Click()
End
End Sub
```

This file (IfThen.txt), can be downloaded from the course web site.

Code 6.9

Note that this program has an extra procedure - *Sub EndProgram_Click()*. You have probably found that Visual Basic programs do not terminate by default. By adding an extra button on the form and programming this with '*End*' between the lines of code, this will terminate the program.

This will be used on all subsequent programs.

Sample output is shown in Figure 6.11.

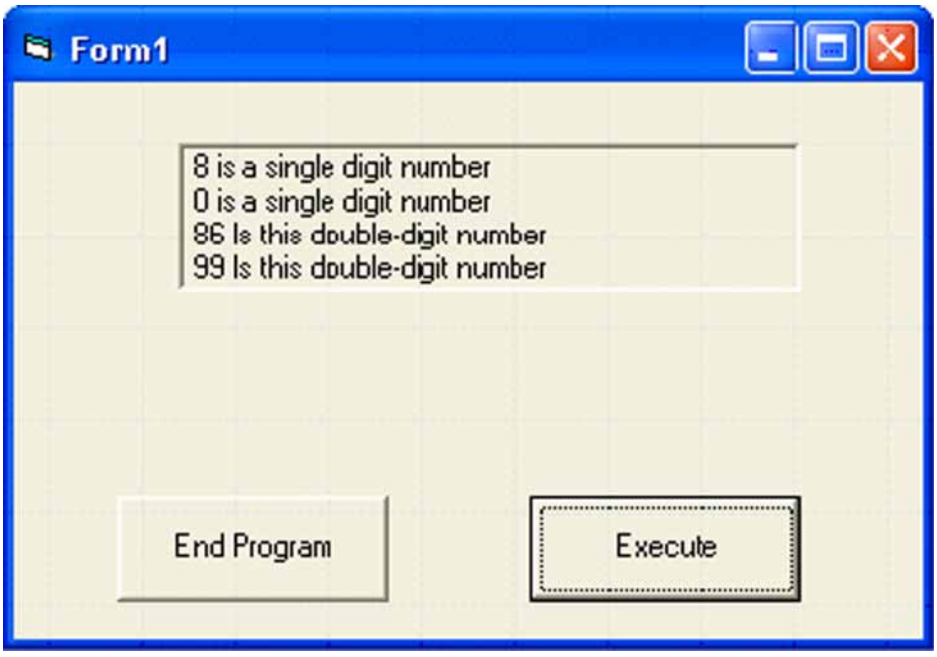


Figure 6.11:

Further examples

Example 1

Problem: What is the output of the following code?

```
Dim Number1 As Integer, Number2 As Integer

  Number1 = 10
  Number2 = 5
  if Number1 > Number2 then
    Print "Number1 is biggest"
  else
    Print "Number2 is biggest"
  End if
```

Solution: This piece of code first declares two integer variables, `Number1` and `Number2`. `Number1` is assigned a value of 10, and `Number2` is assigned a value of 5.

The condition for the `if` statement is "is `Number1` greater than `Number2`"? If this is true then the statement "Number1 is biggest" is displayed. If the condition is false then the statement "Number2 is biggest" is displayed on the screen.

In this case the condition is true since `Number1` has a value of 10, which is greater than the value of 5 that has been assigned to `Number2` and so the output from this code is

```
Number1 is biggest
```

Example 2

Problem: What is the output of the following code?

```
Dim Value1 As Integer, Value2 As integer

  Value1 = 10
  Value2 = 0
  if (Value1 <> 0) and (Value2 <> 0) then
    Print "One"
  else
    Print "Two"
  End if
```

Solution: This condition for the `if` statement uses a logical and. It is testing if `Value1` is not equal to zero and if `Value2` is not equal to zero. In this case `Value1` is equal to zero (true) but `Value2` is not (false). Therefore the whole expression is false, and the text Two would be printed on the screen.

30 min

Which is bigger?

Write a program which prompts a user to type in two numbers - first one, then the other. The program responds by printing the bigger of the two. Use the code fragments in the examples above to help you, and make sure that the program has user-friendly prompts for the input and output.

Even or odd?

30 min

Write a program to test if a user entered number is even or odd.

In this program you will have to use the modulus operator (`mod`) to calculate the remainder when the value stored in the variable `number` is divided by 2. An even number has no remainder if divided by 2.

Calculating wages

20 min

Write a program to calculate the commission based wages of a computer salesman. His basic wage is £50 per week and he is expected to sell at least 10 computers. If he sells more than 10 computers, he receives an extra £5.50 per computer he sells after the 10th computer.

The basic algorithm could be drawn as shown in Figure 6.12

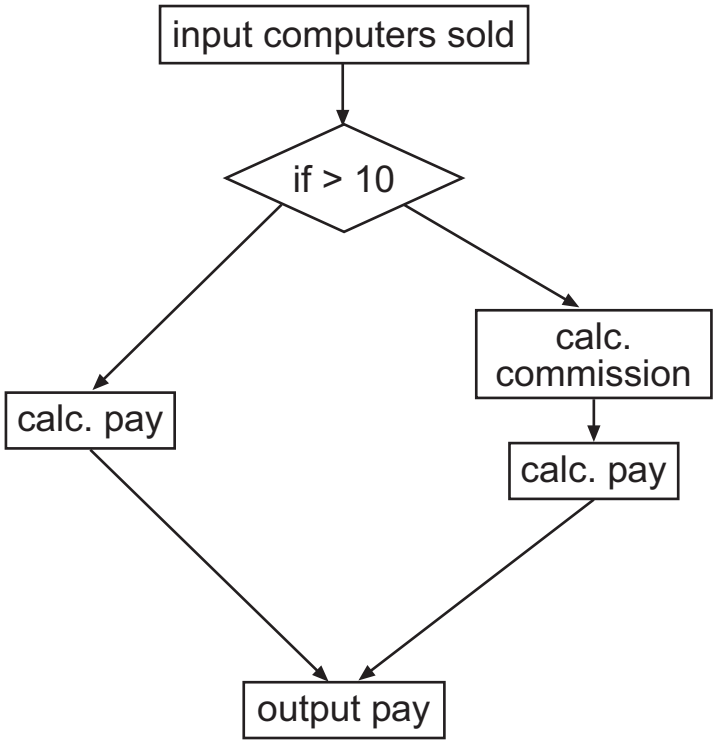


Figure 6.12

Outwith range

When does the following statement evaluate to be TRUE?

```
(number < 0) or (number > 9)
```

6.13.1 Comparison Operators

There are two kinds of operators that allow comparison within Visual Basic:

1. Relational
2. Logical

6.13.2 Relational operators

Relational operators are used for testing of conditions. They are used to construct the 'expression' which is used in the `if` statement. The relational operators check whether two quantities are the same or whether there is a difference between them. Because of the inexact way floating point numbers are stored in computers you should *not* use the equality operator = between two real numbers. Even if two real numbers are printed to the screen as the same two numbers, this is no guarantee that they are the same internally.

The only way to test equality between real numbers A and B is to use the following expression:

```
If ABS(A - B) < small difference Then.....
```

The function `ABS` returns the absolute value of any number.`ABS(5.3)` gives 5, `ABS(-43)` gives 43

Depending upon how accurate real numbers are held on the computer small difference is usually of the order 0.0000001.

The relational operators are summarised in Table 6.9. These operations can be performed between most types of object, but they will be used most often to compare two integer values or two characters.

Table 6.9: Relational Operators

Symbol	Example	Meaning
=	`a = b`	`a` equal to `b`
<	`a < b`	`a` less than `b`
<=	`a <= b`	`a` less than or equal to `b`
>	`a > b`	`a` greater than `b`
>=	`a >= b`	`a` greater than or equal to `b`
<>	`a <> b`	`a` not equal to `b`

Example 1 - Using a Relational Operator

Problem: How can a comparison be made between two variables, `Num1` and `Num2` to find out if `Num1` is greater than or equal to `Num2`?

Solution: The comparison can be made using the relational operator >=, e.g.

```
Num1 >= Num2
```

This comparison could be reversed to check if `Num2` is less than `Num1`, e.g.

```
Num2 <= Num1
```

Example 2 - Using a relational operator in an `if` statement

Problem: Construct an if statement that will display a message to the screen if two variables, `Val1` and `Val2`, are equal.

Solution: A typical solution is shown here.

```
Dim Val1 As Integer, Val2 As integer
  ...
  if Val1 = Val2 then
    Print "Val1 and Val2 are equal"
  end if
```

Sentence completion - relational operators

On the Web is a interactivity. You should now complete this task.

6.13.3 Logical Operators

Logical operators test conditions as either being *true* or *false*. In computer programming they are referred to as Boolean operators, named after George Boole, a mathematician and logician.

6.13.4 Logical AND

Logical `and` means that an expression is true only if both the part before the '`and`' and the part after the '`and`' are true. Both must evaluate to true . So...

`if (number < 0) and (number > 10) then`

the expression will always evaluate to `false`. Why? Because a number cannot be less than zero and greater than 10 at the same time, and both must be `true` for the '`and`' statement to produce a `true` result.

Use of the logical '`and`' operator can be seen in the expression below:

```
if (Vone = 6) and (Vtwo = 6) then
  statement 1
      else
  statement 2
End if
```

This code means that when both `Vone` and `Vtwo` are equal to 6 then `statement 1` is executed. If either or both of the conditions is `false` then `statement 1` is ignored and execution passes onto `statement 2`.

Example 1 - Passwords

Problem: A program is written which asks the user to enter a password and to confirm it by entering it again. If both words are equal to a program constant then the program will display a suitable comment. If not, or only one instance of the password is entered the user will be informed.

The algorithm is shown below:

Solution:

```
1. request 1st  password from the user
2. request 2nd password from the user
3. if (the first password = constant) and (the second password = constant)  then
4.   display the word entered both times
5. else
6.   display the two words and that they are different"
```

The full Visual Basic code for this program is as follows:

```
Option Explicit
Private Sub Command1_Click()

  'program using the AND operator

  '15th February 2004
  'Program by Fred Fink

  'This program will prompt the user to enter two words
  'and compare them to a string constant

  Dim Word1 As String
  Dim Word2 As String

  Const pass As String = "Daffodil"

  Word1 = InputBox("Please input password ")
  Word2 = InputBox("Password again, please")

  If (Word1 = pass And Word2 = pass) Then
    PicDisplay.Print "Both words "; pass; " are identical"
       Else
    PicDisplay.Print Word1; " and "; Word2; " are not identical"
  End If

End Sub


Private Sub Command2_Click()
End
End Sub
```

This file (Password.txt), can be downloaded from the course web site.

Code 6.10

Program output is seen in Figure 6.13

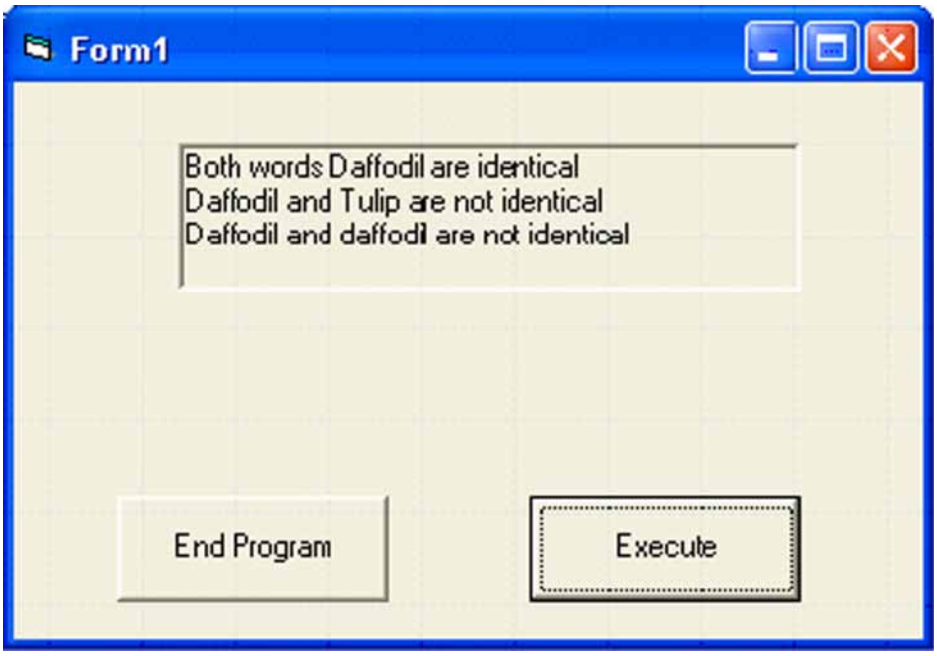


Figure 6.13:

Notice from the third entry that the input is case sensitive.

How could this be resolved?

Using the logical `and` operator in an `if` statement

Construct an `if` statement that displays a message if both the variables, `number1` and `number2` are greater than 10.

Validating numeric input and program testing

20 min

Use this fragments of code you have just seen in this section to help you write a program which will accept a value from the keyboard and print 'yes' if it is positive and even. Test your program with positive and negative numbers, odd and even numbers and make sure it passes all the tests. Make a tabulated list of the numbers you use and the results the program gives for each input. Use both positive and negative numbers, not neglecting zero.

6.13.5 The Logical OR (Inclusive)

If either the expression before the `or` or after the `or` is `true`, then the whole expression is true. This also means that if both the expressions are true, the `or` expression will evaluate to `true`.. So or means 'either one, or the other or both'. Hence the term *inclusive*.

Use of the logical OR operator is shown below:

```
if (Vone = 6) or (Vtwo = 6) then
  statement 1
else
  statement 2
end if
```

This code means that when either `Vone` or `Vtwo` is equal to 6 then `statement 1` is executed. If both of the conditions are false then `statement 1` is ignored and execution passes onto `statement 2`.

Example 1 - Wash the car?

Problem: A program is written to prompt the user to enter two conditions - a temperature and a weather forecast. If either or both of the conditions are met then the program will display a suitable message to go and wash the car. If no conditions are met the user will be informed that washing the car is not a good idea. The conditions are compared to program constants.

The algorithm is shown below:

Solution:

```
1. request a temperature from the user
2. request weather condition from user
3. if (the first input >= 15) or (the second input = "Sunny") then
4.   display "Wash the car!"
5. else
6.   display "Not a good idea!"
```

The Visual Basic code for this program is shown in Code 6.11

```
Option Explicit

Private Sub Command1_Click()

  'program using the OR operator


  '15th February 2004
  'Program by Fred Fink

  'This program will prompt the user to enter a value and
  'a condition

  Dim iTemp As Integer
  Dim weather As String

  iTemp = InputBox("Please input today's temperature ")
```

```
    weather = InputBox("Please input Sunny or Rainy")

    If (iTemp >= 15 Or weather = "Sunny") Then
      PicDisplay.Print iTemp; "deg "; " and "; weather; "! Wash the car"
         Else
      PicDisplay.Print iTemp; "deg "; " and "; weather; "! No way!"
    End If

End Sub


Private Sub Command2_Click()
End
End Sub
```

This file (WashCar.txt), can be downloaded from the course web site.

Code 6.11

The program output is shown in Figure 6.15.

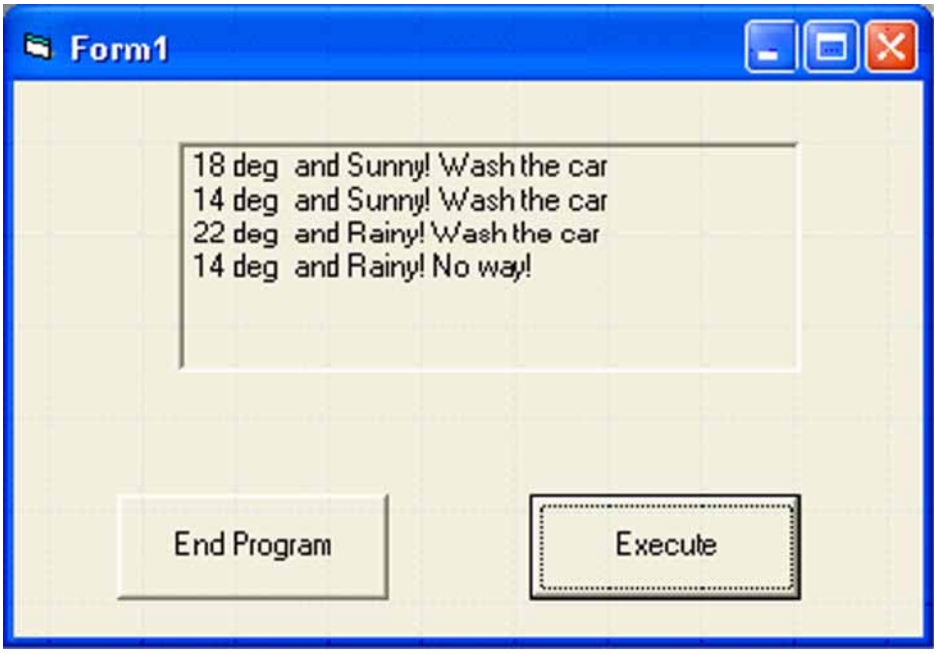


Figure 6.14:

6.13.6 Logical XOR (Exclusive OR)

If either the expression before the `or` or after the `or` is `true`, then the whole expression is `true`. in this case, however if both the expressions are true, the `or` expression will evaluate to `false`. So or means 'either one, or the other but not both'. Hence the term *exclusive*.

Use of the logical `XOR` operator is shown below:

```
if (Vone = 6) xor (Vtwo = 6) then
  statement 1
else
  statement 2
end if
```

This code means that when either `Vone` or `Vtwo` is equal to 6 then `statement 1` is executed. If both of the conditions are true or if both of the conditions are false then `statement 1` is ignored and `statement 2` is executed.

Example The '*Car wash*' program could easily be modified to include the `XOR` instead of `OR`. Rewrite the code and run the program.

6.13.7 Logical Not

Use of the logical `not` operator is shown below:

```
if (Vone = 6) and not (Vtwo = 6) then
  statement 1
end if
```

In this example if `Vone` is equal to 6 and `Vtwo` is *not equal* to 6 then `statement 1` is executed. The `not` operator inverts true/false values e.g. if the value is `true` then `not(true) = false`.

Complete the following table by dragging the TRUE/FALSE boxes into their correct place

Example The '*Word1 and Word2*' program could easily me modified to include the NOT operator. Rewrite the code and run the program

6.13.8 Review Questions

Q17: Operators are used within expressions to assign values to variables and perform calculations. Which one of the following operators has the highest precedence in Visual Basic?

a) +
b) AND
c) /
d) NOT

Q18: What would be the result of evaluating the following expression 3 + 4 * (6 - 4)?

a) 11
b) 14
c) 23
d) 38

Q19: Which one of the following describes **sequencing**?

a) Alteration in the flow of control based upon the test of a condition
b) The repetitive execution of a sequence of instructions
c) The execution of program statements in order, from beginning to end
d) Exiting a loop structure after a sequence of instructions

Q20: Which one of the following expressions represents the logical AND operator?

a) Requires only one condition to be tested
b) The expression is TRUE when both conditions being tested are TRUE
c) The expression is TRUE if either the conditions being tested are TRUE
d) Can be used to test for values outwith a given range

Q21: Which one of the following is NOT a control structure in programming?

a) Sequence
b) Repetition
c) Assignment
d) Selection

6.14 Nested IF Statements

If a condition has to be tested that depends on whether another condition is already `True` (such as "If it's 6:30 p.m. and if I'm logged on to SCHOLAR" then.....), nested If statements can be used.

A nested If statement is one that's enclosed within another If statement.

The format for a nested If statement is as follows:

```
If condition Then
   If another_condition Then
       statement
   Else
       another statement
   End If
End If
```

`if` statements can be nested, but care should always be taken to ensure that the `else` statement is associated with the correct `if`.

The following examples exemplify the nested *If* statement:

Example 1 - Testing for range and the number of digits in a number

Problem: Describe the operation of the following program as seen in Code 6.12:

```
Option Explicit

Private Sub Command1_Click()
```

```
'program nested_IF

'15th February 2004
'Program by Fred Fink

'This program will prompt the user to enter a number between 1
'and 99
'The program will test the number to see if it is
'less than 10 and print the message that it is a single-digit
'number.
'If it is > 9 it will output "double-integer number"
'otherwise "out of range"

Dim testNumber As Integer

testNumber = InputBox("Please input a number between 1 and 99")

If (testNumber >= 0) And (testNumber <= 99) Then
  If testNumber < 10 Then
    PicDisplay.Print testNumber; "is a single digit number"
    Else
       PicDisplay.Print testNumber; "is a double-digit number"
  End If
    Else
    PicDisplay.Print testNumber; "is out of range"
End If

End Sub

Private Sub ExitProgram_Click()
End
End Sub
```

This file (NestedIF.txt), can be downloaded from the course web site.

Code 6.12

Solution: One variable, `testNumber`, is declared as an `integer`. The user is prompted to enter a number between 1 and 99 and the value entered is stored in the variable `number`. If the value is not within the specified range (greater than 0 and less than 100), then the `else` part of the outer `if` statement is executed and the message *Number is out of range* is displayed on the screen and the program is finished.

On the other hand, if the value is within range, then the first part of the outer `if` statement is executed which contains several lines of code between `if` and `End If`. Within this block of code is another `if` statement which tests if the number is less than 10, in which case the message *single digit number* is displayed on the screen, otherwise (else), the message *double digit number* is displayed on the screen.

The program output is shown in Figure 6.15

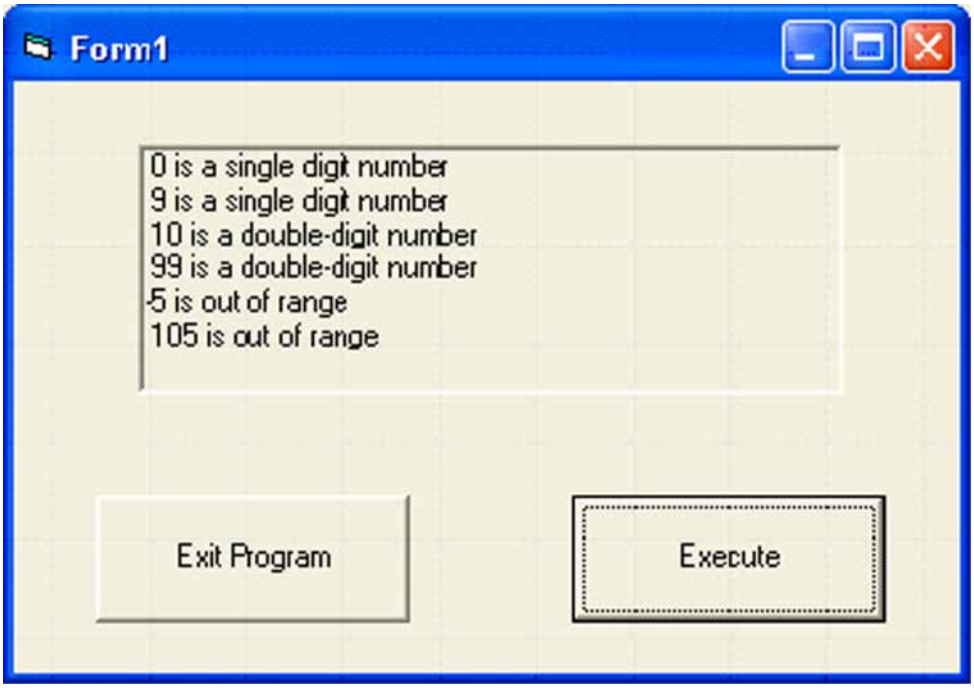


Figure 6.15:

Please take time to follow this explanation through. If you can see the 'how' and 'why' of it, you will be able to program `if` statements - nested or plain - without any worries

6.15 If...Then...ElseIf

Nested `if` statements can be fairly complex to follow and can give rise to what is known as '*spaghetti code*'

The `ElseIf` statement can simplify expressions to a certain extent. The structure of the statement is:

```
if expression1 then
  statements 1
elseif expression2 then
  statements 2
elseif expression3 then
  statements 3
else
  statements 4
```

These expressions are evaluated in order, if any expression is true, the statement associated with it is executed and this terminates the whole chain. As many decisions as desired may be included within the chain. As always, the code for each statement is either a single statement or a group of statements.

Consider the situation where you have two boolean variables, `booly1` and `booly2`, and you need to take actions for each of the possible combinations of true and false.

Using nested `if` statements would produce:

```
If booly1 = True Then
    If booly2 = true Then
         statements for true/true
    Else
         statements for true/false
    End If
Else
    If booly2 = false Then
         statements for false/true
    Else
         statements for false/false
    End If
End If
```

Using `ElseIF` will produce the following, more compact code:

```
If booly1 And booly2 Then
     statements for true/true
ElseIf booly1 And Not booly2 Then
     statements for true/false
ElseIf Not booly1 And booly2 Then
     statements for false/true
Else
     statements for false/false
End If
```

The following examples will show the `If..EndIf` statement in use:

Example 1 - Using an elseif statement to calculate the number of digits in a number

Problem: How can an `else if` statement be used to find out whether an integer number is within a certain range (a positive number) and how many digits it contains?

Solution: A typical solution to this problem is shown here.

```
if number >= 1000 then
  n_digits = 4
elseif number >= 100 then
  n_digits = 3
elseif number >= 10 then
  n_digits = 2
elseif number >= 0 then
  n_digits = 1
else
  print("Value out of range")
```

The above code shows an example of how this could be achieved.

- You start testing if the variable `number` is greater than or equal to 1000, in which case there are 4 digits (you must be assuming that the variable `number` is not greater than 999) and the variable `n_digits` is set to 4. If it is not greater than or equal to 1000, then you move onto the next branch of the `if` statement.
- The next condition tests the variable `number` to see if it is greater than or equal to 100. If this is true then you know the variable `number` has a value between 100 and 999, and so it must have 3 digits, and the variable `n_digits` is set to 3. Again, if this condition is not fulfilled, then you move onto the next branch of the `if` statement.
- The next condition tests the variable `number` to see if it is greater than or equal to 10. If this is true then you know the variable `number` has a value between 10 and 99, and so it must have 2 digits, and the variable `n_digits` is set to 2. Again, if this condition is not fulfilled, then you move onto the next branch of the `if` statement.
- The next condition tests the variable `number` to see if it greater than or equal to 0. If this is true then you know that the variable number has a value between 0 and 9, and so it must have one digit. The variable `n_digits` is set to 1.
- The next branch of the `if` statement is not an `elseif` branch, but an `else` branch, and so does not have a condition but is the branch that is executed if none of the other conditions are met. Execution of this branch of the `if` statement results in the message `Value out of range` being displayed on the screen.

Example 2 - Grades and Marks

Problem: Write a program using the `ElseIf` construct for a user to input a test result and output the corresponding grade. Test scores range from 0 to 100 and the grades from "A" being the highest to "E" the lowest.

Solution

Use the following algorithm:

```
1  ask user to input a mark between 0 and 100
2  If mark (>=0 and < 30) then grade "E"
3    elseIf mark(>=30 and < 49) then grade "D"
4    elseif mark(>=49 and < 65) then grade "C"
5    elseif mark(>=65 and < 85) then grade "B"
6    elseif mark(>=85 and <=100) then grade "A"
7    else number out of range
8  end if
9  If mark (>=0 and <= 100)
10   display grade
11 end if
```

The full Visual Basic program is shown in Code 6.13:

```
Option Explicit
Private Sub Command1_Click()
  'Marks and grades program
  '15th February 2004
  'Program by Fred Fink

  'This program will prompt the user to enter a mark and
  'a grade will be output

  Dim iMark As Integer
  Dim sGrade As String

  iMark = InputBox("Please input a test mark ")

  If (iMark >= 0) And (iMark < 30) Then
    sGrade = "E"
    ElseIf (iMark >= 30) And (iMark < 50) Then
    sGrade = "D"
    ElseIf (iMark >= 50) And (iMark < 65) Then
    sGrade = "C"
    ElseIf (iMark >= 65) And (iMark < 85) Then
    sGrade = "B"
    ElseIf (iMark >= 85) And (iMark <= 100) Then
    sGrade = "A"
    Else
    PicDisplay.Print iMark; " is out of range! Try again."
  End If

  If (iMark >= 0) And (iMark <= 100) Then
    PicDisplay.Print iMark; " = grade "; sGrade
  End If
End Sub

Private Sub Command2_Click()
End
End Sub
```

This file (Grades.txt), can be downloaded from the course web site.

Code 6.13

Figure 6.16 shows the program output:

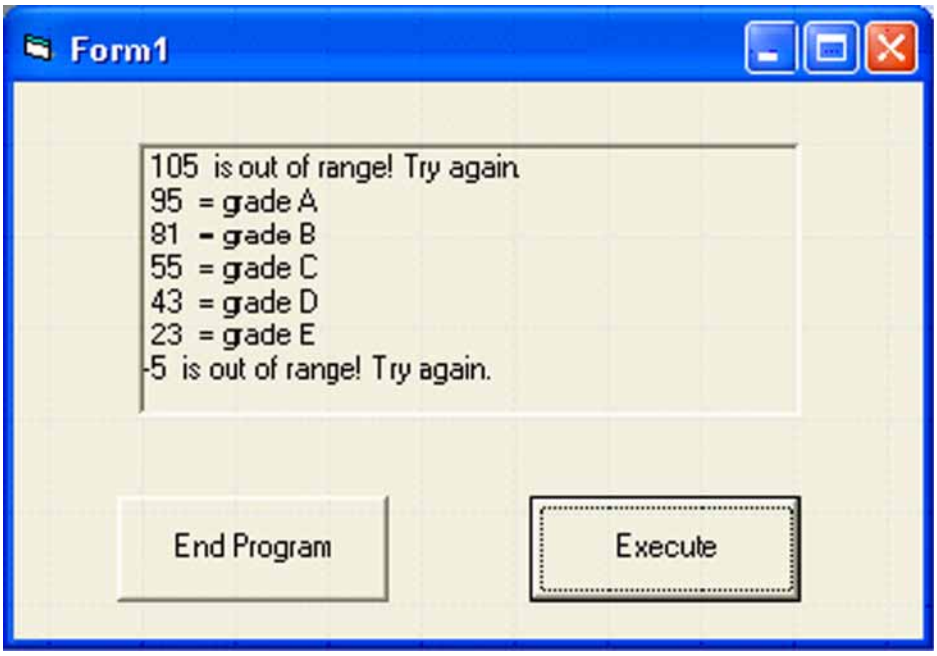


Figure 6.16:

Calculating the number of digits in a number


20 min

Learning Objective

Refining an existing program to make it user-friendly

Write a program which will calculate the number of digits in a number. The error messages should be helpful and user-friendly. The program should deal correctly with numbers having four, or more, digits by giving the error message *Value out of range. Maximum of three digits numbers, please.*

Calculating Leap Years


20 min

Learning Objective

Be able to use the `if` statement.

Be able to make us of constants where appropriate.

A leap year is a year which is exactly divisible by 4, unless it is exactly divisible by 100 in which case it is only a leap year if it is exactly divisible by 400.

Write a program which prompts the user for a year and then displays if it is a leap year or not.

Note: Use `else if` in your code. You will also need to be careful that you do the tests for leap years and leap centuries in the correct order. If you test for leap years first, then 1900 is divisible by 4, but as it is not a leap century it will not be a leap year either.

6.16 The Select Case Statement

The `Case` statement is a special instance of the `If...Then...ElseIf` structure.

To implement multi-way decisions the `case` statement provides a more concise and elegant representation than multiple `else if` and `nested if` statements, which can get very difficult to follow. Probably if you have more than three `else if` statements you should consider using the `case` statement instead.

The `case` statement is particularly useful when selection is based on a single variable or a simple expression. This is called the *case selector*. Note that the case selector must be an ordinal data type i.e. one whose values can be listed, such as `integer`, `string` or `boolean`.

The structure of the `Case` statement is:

```
Select case test expression

Case value1
       Block of one or more statements
Case value2
       Block of one or more statements
Case value3
       Block of one or more statements
Case Else
       Final block of one or more statements

End Select
```

The first statement of the `Select Case` block is the *Select Case* statement itself. This statement identifies the value to be tested against all possible results. This value, represented by the `Value 1`, `Value 2` etc. and can be any valid numeric or string expression, a variable, a logical expression, or a function.

Each group of commands is initiated by the `Case` statement. The `Case` statement identifies the expression to which the `Value` is compared. The `Case` statement can express a single value or a range of values. If the `Value` is equal to or within range of the expression, the commands after the `Case` statement are run. The program runs the commands between the current `Case` statement and the next `Case` statement or the `End Select` statement. If the `Value` isn't equal to the value expression or doesn't fall within a range defined for the `Case` statement, the program proceeds to the next `Case` statement.

A number of worked solutions to problems are given here - make sure you understand what is going on in these examples. Run these programs for yourself and experiment with making changes to the code to ensure you understand what is happening.

Example 1: Write a program to input an integer between 1 and 7 and the day of the week will be output.

Solution

Use the following algorithm:

```
1  ask user to input number between 1 and 7 inclusive
2  select case number
3  case is = 1 output "Sunday"
4  case is = 2 output "Monday"
5  case is = 3 output "Tuesday"
6  case is = 4 output "Wednesday"
7  case is = 5 output "Thursday"
8  case is = 6 output "Friday"
9  case else output "Saturday"
10 end select
```

Note the use of keyword `is`. Whenever a condition is expressed within a statement the keyword `is` is used to impose the condition.

The full Visual Basic program is shown in Code 6.14

```
Option Explicit
Private Sub Command1_Click()
  'Use of the Case Select statement(1)
  '15th February 2004
  'Program by Fred Fink

  'This program will prompt the user to enter a number
  'and the day will be output

  Dim iDay As Integer

  iDay = InputBox("Please input a number between 1 and 7 ")

  Select Case iDay
    Case Is = 1
    PicDisplay.Print iDay; " =  Sunday"
    Case Is = 2
    PicDisplay.Print iDay; " =  Monday"
    Case Is = 3
    PicDisplay.Print iDay; " =  Tuesday"
    Case Is = 4
    PicDisplay.Print iDay; " =  Wednesday"
    Case Is = 5
    PicDisplay.Print iDay; " =  Thursday"
    Case Is = 6
    PicDisplay.Print iDay; " =  Friday"
    Case Else
    PicDisplay.Print iDay; " =  Saturday"
```

```
  End Select
End Sub

Private Sub Command2_Click()
End
End Sub
```

This file (Case1.txt), can be downloaded from the course web site.

Code 6.14

The program output is shown in Figure 6.17

Figure 6.17:

Example 2 - Using the `case select` statement with a range of values

Problem: Rewrite the previous `If..Then..ElseIf` program, Code 6.13, using `Select Case` Statement

Solution

Here the keyword `To` is used to represent values within a range.

Use the following algorithm:

```
1  ask user to input a mark between 0 and 100
2  select case mark
3  case 85 to 100 output Grade 'A'
4  case 65 to  84 output Grade 'B'
5  case 50 to  64 output Grade 'C'
6  case 30 to  49 output Grade 'D'
7  case  0 to  29 output Grade 'E'
8  case else output "mark outside range! Try again"
9  end select
```

The full Visual Basic program is shown in Code 6.15

```
Option Explicit
Private Sub Command1_Click()
'Use of the Case Select statement
'15th February 2004
'Program by Fred Fink

'This program will prompt the user to enter a date
'and the month will be output

Dim iMark As Integer

iMark = InputBox("Please input a test mark ")

Select Case iMark
    Case 85 To 100
    PicDisplay.Print iMark; " = grade A "
    Case 65 To 84
    PicDisplay.Print iMark; " = grade B "
    Case 50 To 64
    PicDisplay.Print iMark; " = grade C "
    Case 30 To 49
    PicDisplay.Print iMark; " = grade D "
    Case 0 To 29
    PicDisplay.Print iMark; " = grade E "
    Case Else
    PicDisplay.Print iMark; " is out of range! Try again."
End Select
End Sub

Private Sub Command2_Click()
End
End Sub
```

This file (Case2.txt), can be downloaded from the course web site.

Code 6.15

Program output is shown in Figure 6.18:

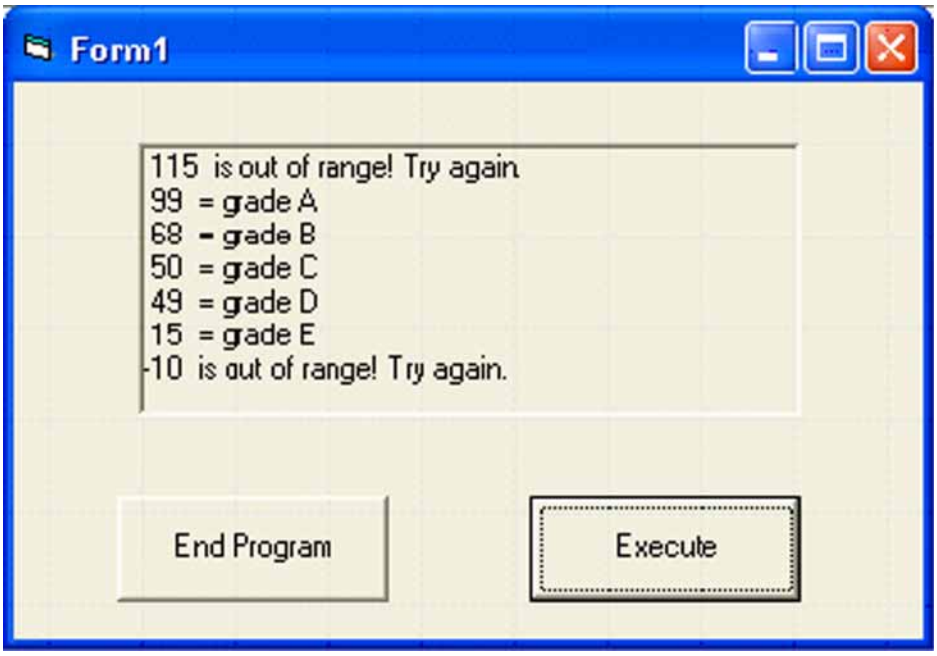


Figure 6.18:

Example 3 - Extending the testing for a weekend or weekday using text Input

Problem: Write a program to test whether a `string` variable `Day` holds text that represents a week day or a weekend. Assume input can be in either lower or upper case.

Solution: The statement in the previous example, Code 6.14, could be extended to give the following program Code 6.16:

```
Option Explicit
Private Sub Command1_Click()

  'program use of case statement (3)
  '16th February 2004
  'Program by Fred Fink

  'To show the use of case with upper or lower case input

  Dim Day As String
  Dim FirstLetter As String

  Day = InputBox("Please enter the day")
  FirstLetter = Left$(Day, 1)     'Pick off first character

  Select Case FirstLetter
    Case "s", "S"
    PicDisplay.Print Day; " is the weekend; "
    Case "m", "M"
    PicDisplay.Print Day; " is a weekday "
    Case "t", "T"
    PicDisplay.Print Day; " is a weekday "
```

```
        Case "w", "W"
        PicDisplay.Print Day; " is a weekday "
        Case "f", "F"
        PicDisplay.Print Day; " is a weekday "
        Case Else
        PicDisplay.Print Day; " is not a day! "
      End Select

    End Sub

    Private Sub Command2_Click()
    End
    End Sub
```

This file (Case3.txt), can be downloaded from the course web site.

Code 6.16

Program output is shown in Figure 6.19:

Figure 6.19:

It is really important that you study this example carefully. You will probably use the `case` statement a lot.

This program uses the `case` statement to check whether the letter typed in by the user matches `s` (Saturday and Sunday), `m` (Monday), `t` (Tuesday and Thursday), `w` (Wednesday) or `f` (Friday). If the match is with `s` it can only be a weekend. All other matches must be week days.

Note: Here you are comparing with the single character and not the entire string called Day so single quotes are required to show that you are using a character.

In order to test the first character of the string `Day` the Visual Basic in-built function `Left$` is used.

For example `Left$(Monday, 1)` would produce the character `"M"`

Sentence completion - `case` statement

On the Web is a interactivity. You should now complete this task.

40 min

Musical Notes

Learning Objective

Be able to use the `switch` statement.

a, b, c, d, e, f and g are valid musical notes.

Write a program that allows the user to input a character and uses a `CASE` statement to test and display if the character is a valid musical note.

6.17 Summary

The following summary points are related to the learning objectives in the topic introduction:

- introduction to the Visual Basic environment;
- existence of various data types including local and global forms;
- what is meant by the scope of a variable;
- discussion of formatted input/output with coded examples;
- discussion of conditional statement structures with coded examples;
- exemplification of the CASE statement.

6.18 End of topic test

An online assessment is provided to help you review this topic.

Topic 7

High Level Language Constructs 2

Contents

Prerequisite knowledge

Before studying this topic you should be able to describe and use the following constructs in pseudo-code and a suitable high level language:

- *fixed loops;*
- *conditional loops using simple and complex conditions;*
- *nested loops;*
- *1-D arrays.*

Learning Objectives

- *Be able to declare a 1-dimensional array*
- *Be able to access array elements*
- *Be able to manipulate data within arrays using iterative structures*

Revision

Q1: An array is described as a structured data type. This means that:

a) Data items are all in order
b) Data items will take up a lot of computer memory
c) Data items of the same type are grouped together
d) None of the above

Q2: What occurs when a 1-D array is initialised?

a) All array elements are set to the same value
b) All array elements are put into order
c) Only the first array element is set to some value
d) Nothing occurs when an array is initialised

Q3: A 1-D array string called Days(0-6) hold the days of the week. The 4th array element is assigned the value Wednesday. The correct statement for this is:

a) Days(4) = "Wednesday"
b) Days(3) = Wednesday
c) Days(4) = Wednesday
d) Days(3) = "Wednesday"

Q4: An array called List(5) contains the integers 1 to 6 in sequence. If the 2nd and 4th elements are now assigned the values 8 and 9 respectively, the array List will now contain:

a) 1, 2, 3, 8, 5, 9
b) 1, 2, 8, 4, 9, 6
c) 1, 8, 3, 9, 5, 6
d) 8, 2, 9, 4, 5, 6

Q5: All array elements have to be set with similar values. The process to accomplish this is called:

a) Sequence
b) Selection
c) Iteration
d) None of the above

7.1 Introduction

An important aspect of this topic is the 1-dimensional array. Declaring and initialising arrays are introduced together with how data is manipulated within arrays using various looping structures. Each sub topic has working solutions to the example programs thus providing a suitable environment for building confidence in writing programs before the final topic, dealing with standard algorithms is covered.

7.2 Iteration

The aim of this topic is to introduce you to *iterative* structures, or loops. Iteration simply means repetition, which in the context of programming is the execution of blocks of code many times over.

Iteration is a fundamental part of almost every program and is one of the most useful features of programming. You do not want a computer to produce one payslip, but many payslips; to add up just two numbers but thousands of numbers; to put in order just two items but thousands of items.

There are three different looping constructs you can use in Visual Basic:

1. For...Next loop
2. Do...While loop
3. Do...Until loop

Most loops have the following characteristics in common:

- initialisation
- a condition which evaluates either to TRUE or FALSE
- a counter that increments or decrements by discrete values.

7.2.1 The For..Next loop

The `for` keyword marks the beginning of the code which will be repeated according to the conditions supplied following the `for`.

When *incrementing*, the general form of the statement is:

```
for counter = initial value to final value step value

    statements

Next counter
```

When *decrementing*, the general form of the statement is:

```
for counter = initial value to final value step -ve value

    statements;

Next counter
```

Note:

1. The *initialisation* statement is carried out only once when the loop is first entered i.e. initialise counter to *initial value*
2. The *condition* is tested before each run through the body of the loop. The first test is immediately after initialisation, so if the test fails the statements in the body are not run. An incrementing loop terminates when *counter* $>$ *final*, while a decrementing loop terminates when *counter* $<$ *final*
3. An increment or decrement of the *counter* variable is executed after the loop body and before the next test. The value of the counter is incremented or decremented by the *step value*.
4. the value of *counter* must not be changed in any statements within the body of the loop
5. changing the value of *final* within the loop will have no *effect* on how many times the loop is executed
6. after the loop has terminated, the value of *counter* is *undefined*
7. counter may be any *ordinal* type e.g. `integer, char`

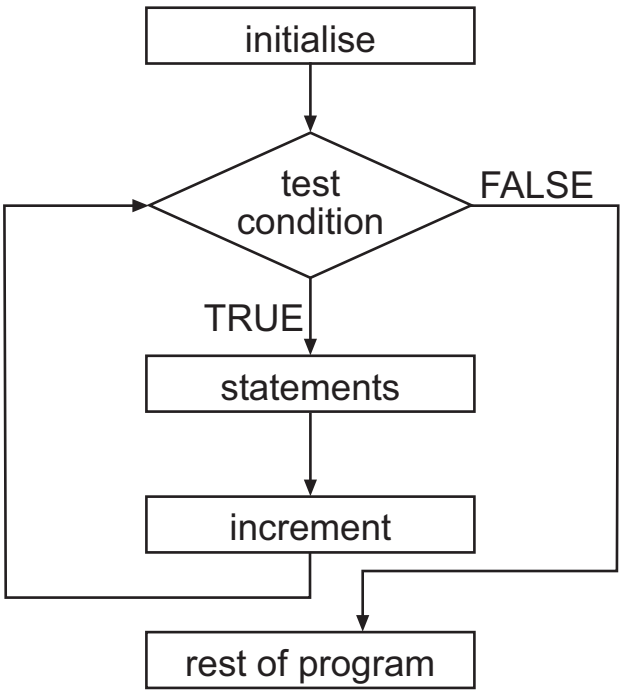


Figure 7.1:

The initial and final states may be *constants, variables* or *expressions* e.g.

```
for counter = (min+7) to (max-5)
```

The following examples will show the `for..next` loop in operation.

Example 1: Converting inches to centimetres

Problem: Write a program that will convert inches to centimetres for a range of values and using an increment of 5. Use the `for..loop` and output the results in tabular form. Use the conversion factor 2.54 centimetres to the inch.

Solution

The algorithm is shown below:

```
2   for each value from 1 to 50 step 5
3   calculate conversion to centimetres using the factor 2.54
4   tabulate output
5   next counter
```

The full Visual Basic program is seen in Code 7.1

```
Option Explicit

Private Sub Command1_Click()
  '16th February 2004
  'Program by Fred Fink

  'This program will illustrate the For..Next loop

  Dim Inches As Integer
  Dim Cms As Single

  PicDisplay.Print Tab(3); "Inches"; Tab(12); "Centimetres"
  PicDisplay.Print

    For Inches = 0 To 50 Step 5
       Cms = Inches * 2.54
       PicDisplay.Print Tab(5); Inches; Tab(15); Cms
    Next Inches

End Sub
```

Code 7.1

The program output is shown in Figure 7.2

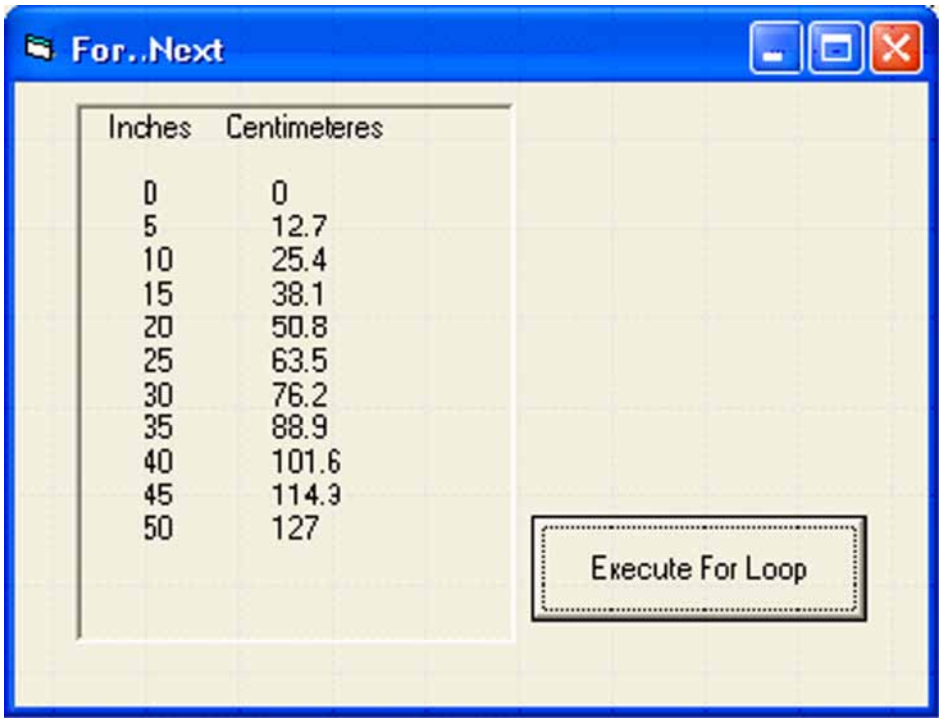


Figure 7.2:

Note that the output of this program is in tabular form. This was achieved using the `TAB()` function, first to output the heading then the results. Tab is short for tabulate and the output items are separated by the values within the tab statements.

If the value of `step` is not specified it is assumed by Visual Basic to be the value 1 by default.

Example 2: Displaying integers using negative step

Problem: Write a program that will output integers from 100 to 65 together with the corresponding characters that the numbers represent. Use a for..next loop with -ve step and output the results in tabular form.

Solution:

```
1. display "The integers from 100 to 65"
 2. for counter = 100 to 65 step -1
 3.    display the value of the integer and character
 4. next counter
```

The Visual Basic program is shown in Code 7.2

```
Option Explicit

Private Sub Command1_Click()
  '16th February 2004
  'Program by Fred Fink

  'This program will illustrate the For..Next loop using -ve step
```

```
    Dim Counter As Integer
    Dim Char As String

    PicDisplay.Print Spc(3); "Integer"; Spc(12); "Character"
    PicDisplay.Print

      For Counter = 90 To 65 Step -1
          Char = Chr$(Counter)
         PicDisplay.Print Spc(5); Counter; Spc(15); Char
      Next Counter

End Sub
```

Code 7.2

Note that, in this case the output has been formatted using the `SPC()` function. This allocates a number of spaces between output items depending on the value expressed within the function.

The program also uses the `CHR$()` function to convert a numerical value to its corresponding character.

For example: `Chr$(65)` returns the character "A"

Part of the program output is shown in Figure 7.3

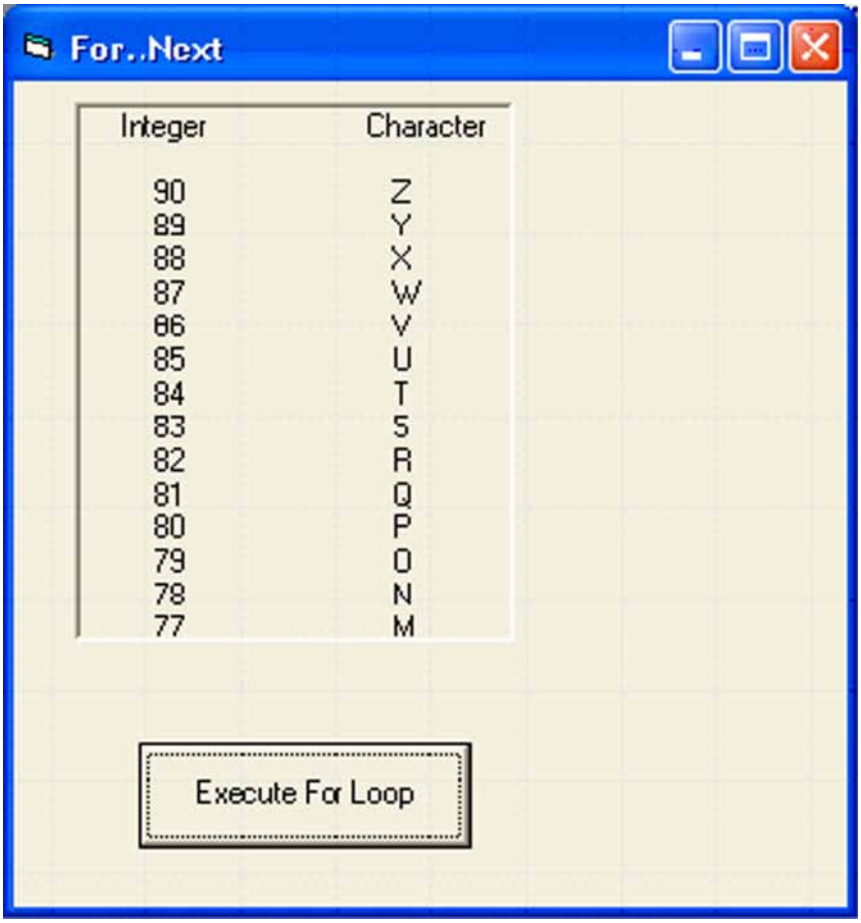


Figure 7.3:

7.2.2 Nested For loops

`For..Next` loops can be nested to allow the programming of loops within loops.

Example 1: Use of nested loops

Consider the Visual Basic program in Code 7.3. See if you can visualise what the output will be before looking at the results screen:

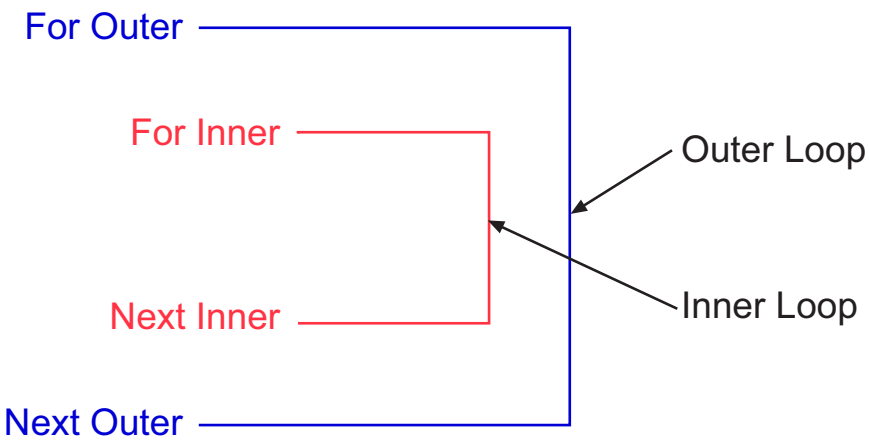
```
Option Explicit

Private Sub Command1_Click()
  '16th February 2004
  'Program by Fred Fink

  'This program will illustrate nested loops

  Dim Outer As Integer, Inner As Integer

    For Outer = 1 To 15
       For Inner = Outer To 15
       PicDisplay.Print Inner;
       Next Inner
```

```
            PicDisplay.Print
        Next Outer

    End Sub
```

This file (NestedIF2.txt), can be downloaded from the course web site.

Code 7.3

In this program there are two loops that are controlled by the variables `Outer` and `Inner`.



The outer loop is initialised with the variable `Outer = 1`. The inner, nested loop is now executed 15 times and the first line of numbers are printed on the same line. This is achieved by using the semi colon at the end of the first print statement. `Outer` then takes on the `value 2` and the process repeats itself until `Outer = 15`. Each time the output is decreased by 1 as the outer loop is executed until the value 15 is reached.

The print statement on its own ensures that a new line is taken for the next row of output. The print statement on its own basically means a line feed.

The output of the program is shown in Figure 7.4:

Note that it is considered bad programming practice to jump out of loops without terminating them fully. After the loops terminate the counter variables are discarded so if this is aborted prematurely, program output may not be as expected.

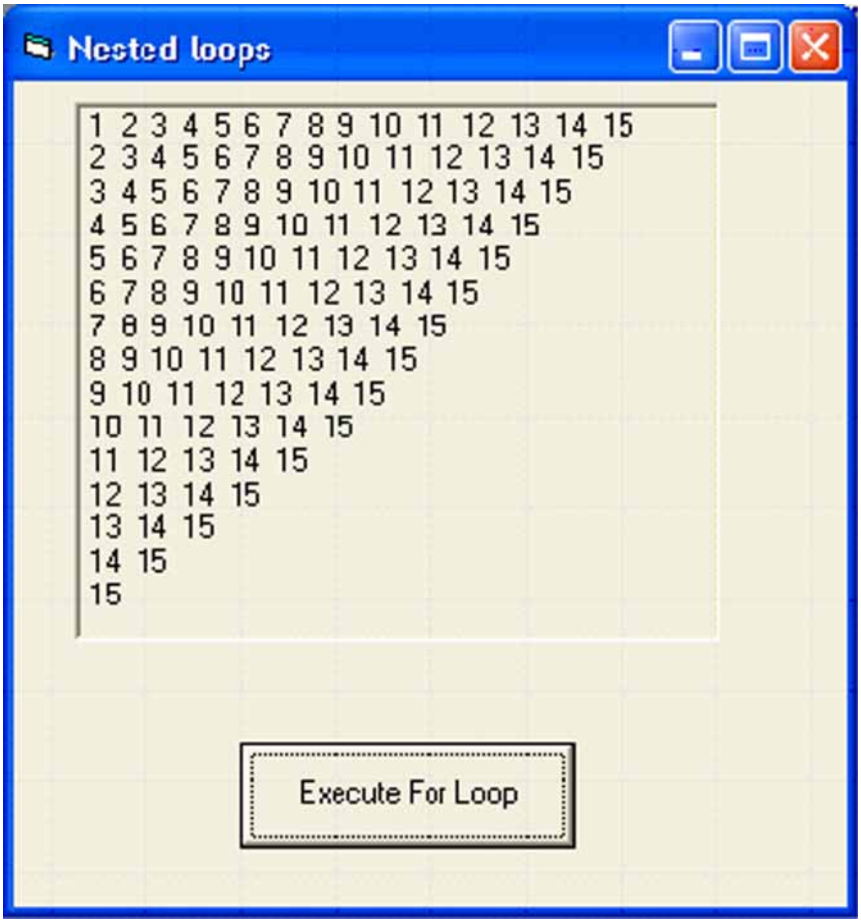


Figure 7.4:

Experiment with the loops and range of numbers to arrive at different outputs.

Problem 2: Use of an If statement with nested For..Next loop

Problem: Write a program that calculates bank interest on a sum of money that is input by the user. Output the capital sum together with the interest.

Solution:

Consider the following algorithm:

```
1  Ask user to input value
2  If value <> 0
3  for count = 1 to 10
4  calculate interest on value
5  output value and interest
6  next count
7  else output message "Value not valid - try again!"
8  end if
```

The Visual Basic Code 7.4 is shown:

```
Option Explicit
Private Sub Command1_Click()
  '16th February 2004
  'Program by Fred Fink

  'This program will illustrate nested loop and IF

  Dim Capital As Integer, Counter As Integer
  Dim Interest As Single

  Capital = InputBox("Please input capital amount")
  SumDisplay.Print Capital
  If Capital <> 0 Then
  PicDisplay.Print
    For Counter = 1 To 10
       Interest = (Capital * Counter) / 100
       PicDisplay.Print Tab(5); "$" & Capital; Tab(12); "  will  gain
                        " & "$" & Interest & "   at " & Counter & "%"

    Next Counter
    Else
    PicDisplay.Print "$" & Capital & " is not a valid input. Please try again!"
  End If
End Sub
```

Code 7.4

The program involves an `If` statement with an embedded `for..next` statement. If the capital sum entered does not meet the initial condition then control will be passed to the `else statement`. If the condition is met then the program will continue and execute the `for..next` statement and output results.

Note the output line which looks rather complex. All it is doing is concatenating the output as a mixture if string and program variables. Concatenation is not just reserved for strings.

Code and run the program. You should end up with output as shown in Figure 7.5

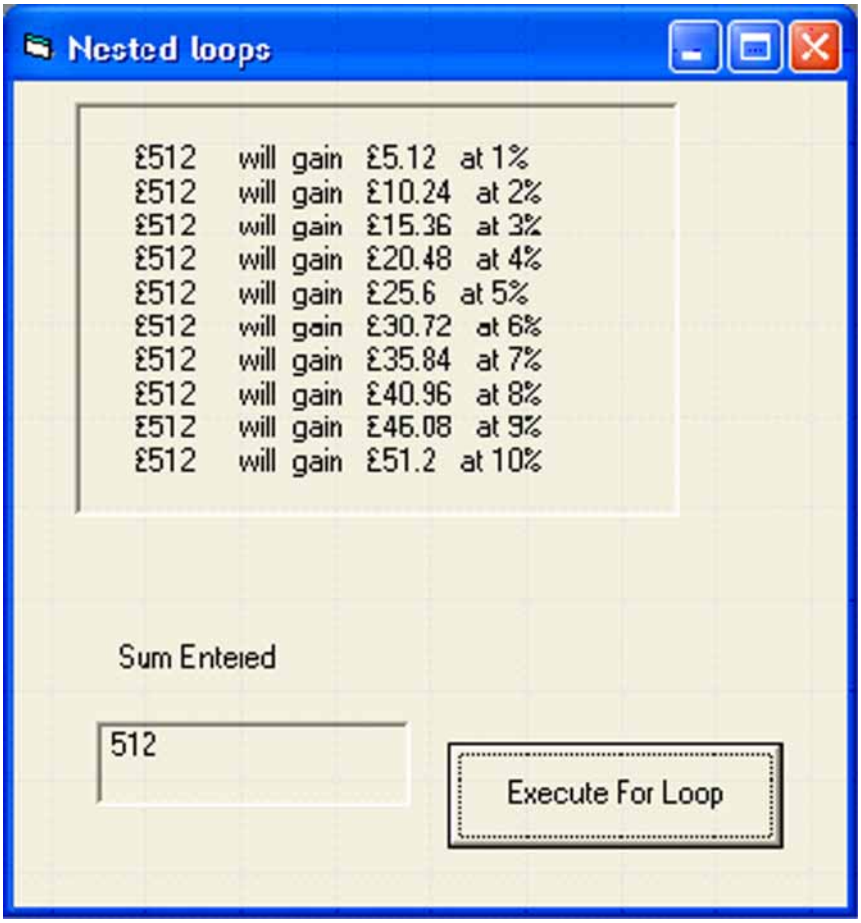


Figure 7.5:

Exercise

Write a program that will output integer values representing the sides of right-angled triangles that satisfy Pythagoras's theorem i.e

```
a^2  +  b^2   =   c^2
```

Only output values that satisfy the equation.

Hint: You will require three nested `for..next` loops and experiment with loop values up to 10 for each loop otherwise the program may run out of memory.

Drawing right-angled triangles

30 min

Use a nested loop to draw right-angled triangle as shown below:

```
*
***
*****
*******
*********
```

30 min

Mowing meadows

Use a nested loop to generate the following lines of a well known verse:

```
1 man went to mow a meadow
2 men, 1 man went to mow a meadow
3 men, 2 men, 1 man went to mow a meadow
...
N men, N-1 men, N-2 men, ....., 1 man went to mow a meadow
```

for N = 10.

7.2.3 Review Questions

Q6: Which one of the following describes correctly an incremental **for** loop?

a) Control variable is decreasing in value by a variable amount
b) Control loop variable is increasing in value by 1
c) Control loop variable is increasing by a variable amount
d) Control loop variable is increasing in value by a constant amount determined by the programmer

Q7: Which one of the following statements is not permitted?

a) For loopCounter = (3*4) to (5*6) step 1
b) For loopCounter = (3*4) to (5*1) step -1
c) For loopCounter = (3*4) to (5*2) step 1
d) For loopCounter = (3*4.16) to (5*5.6) step 1

Q8: Which one of the following problems is best suited to the use of a FOR loop?

a) Calculating the total number of marks entered at a keyboard
b) As an event loop that checks for keyboard input
c) Calculating the average of marks held in a file
d) All of the above

Q9: What shape will be displayed by the following Visual Basic program fragment for any value of N $>$ 1?

```
for i = 1 to N
  for j = 1 to i
     print "+";
  next j
  print
next i
```

a) Triangle
b) Square
c) Rectangle
d) Circle

Q10: What will be displayed by the following Visual Basic program fragment, assuming N = 3

```
sum = 0
for i = 1 to N
    for j = 1 to N
        sum = sum + j
    next j
next i
print sum
```

a) 16
b) 17
c) 18
d) 19

7.3 Formatting output

Up to this point it has been left to Visual Basic to output data in default mode. However it is possible to have more control over how `real values, currency, boolean` variables are output by using the Visual Basic in-built function `Format()`.

The structure of the Format function is:

```
Format (variable, format expression)
```

Table 7.1 shows the formatting functions within Visual Basic:

Table 7.1:

Format name	**Meaning**	**Examples**
General	Displays raw number without separators	12345
Fixed	Displays at least one digit before the decimal point and two digits after the point	67.88
Scientific	Uses scientific notation	6.023 E23
Standard	Displays numbers with separators and two digits after the decimal point	1,234.56
Currency	Same as standard. Negative values are enclosed within parentheses	(1,234.56)
Percent	Displays numbers multiplied by 100 with two digits after the decimal point followed by the % sign	12345.67%

Exercise

Write a Visual Basic program to input a value and output the value in each of the formats in Table 7.1. Use the algorithm below to help you. Make the value large enough so that all aspects of the formatting can be shown.

```
1  Declare variable
2  Input a real value
3  Print "Standard format = ", Format  value, "Standard")
4  repeat for other formats
```

Run the program a few times with different values so that you understand the nature of each format.

7.4 Do Loops

With the `for..next` loop the number of iterations must be known in advance so that the counter variable can be set.

There are many occasions in programming where the number of iterations is unknown so an alternative looping structure has to be used. The `Do...Loop` is a viable alternative to the `for..next` statement.

In Visual Basic, `Do..loops` come in a variety of flavours and for any given program there will probably be more than one solution using a `Do..loop` variant.

There are two main `do..loop` constructs with two variants::

1. `Do While...loop` and `variant Do loop..While`
2. `Do Until...loop` and `variant Do loop..Until`

7.4.1 Do While..loop

The `do while` loop repeats a given set of instructions *while* a given condition is true.

The general form of this statement is:

```
Do while test condition
  statements
Loop
```

The loop repeats itself until the condition becomes `false`.

If the condition is false to begin with then the `do..while` loop will not be entered and control will pass to the rest of the program.

See Figure 7.6

While loops are used in situations where the number of times something has to be repeated is not known. They are often used in data validation where the user is repeatedly asked to enter values when invalid data is detected.

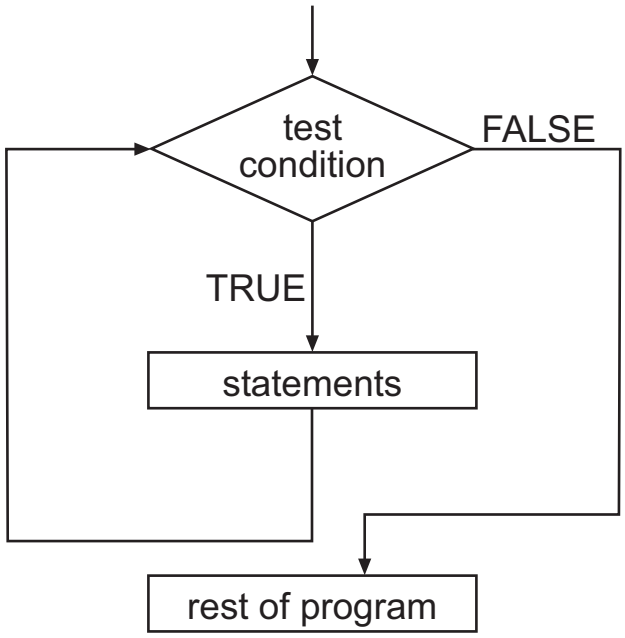


Figure 7.6:

The `do while..loop` is an example of a top tested *while true* structure. The variant do loop..while is an example of a bottom tested *while true* structure:

```
Do
   statements
Loop while test condition
```

The top tested loop can iterate between 0 and N times whereas the bottom tested loop may iterate between 1 and N. This means that using the bottom tested loop the iteration must occur at least once, so the condition can be tested immediately.

Care should be taken when writing programs using loops as an incorrect condition, or mistakes in the body of the loop can cause the program to get stuck in an infinite loop when executing, even when compilation produced no errors. Should this occur in Visual Basic simply click on program `Run` and choose `End`. In more extreme cases pressing `ctrl + alt + delete` will allow you to abort Visual Basic.

7.4.2 Examples of Do While.. Loop

Example 1 - Calculating a sum of positive integers

Problem: A program is written to accept positive numbers from the keyboard, calculate and display a sum of all the numbers entered. It uses a do while..loop which tests whether each number entered is greater than or equal to zero. If the user enters a negative number then the loop will terminate and the total will be displayed on the screen.

Solution:

The algorithm is shown below:

```
1. set the running total to 0
2. display "Give me your first number"
3. get number typed in at the keyboard
4. do while user's number >= 0
5.    add 1 to the running total
6.    display "Give me the next number"
7.    get input typed in at the keyboard
8. loop
9. display the total
```

The Visual Basic program is shown as Code 7.5

```
Option Explicit

Private Sub Command1_Click()
  '17th February 2004
  'Program by Fred Fink

  'This program will illustrate the Do While..loop

  'program addPos

  'This program will add a list of positive numbers typed in
  'at the keyboard. The program will stop when a negative number
  'is entered and display the total

Dim iNumber As Integer, Total As Integer

  Total = 0
  iNumber = InputBox("Give me your first number ")
  Do While iNumber >= 0
      Total = Total + iNumber
      iNumber = InputBox("Give me your Next number")
      PicDisplay.Print ("Your numbers add up to "); Total
  Loop

End Sub
```

This file (DoWhile.txt), can be downloaded from the course web site.

Code 7.5

The program output is shown in Figure 7.7

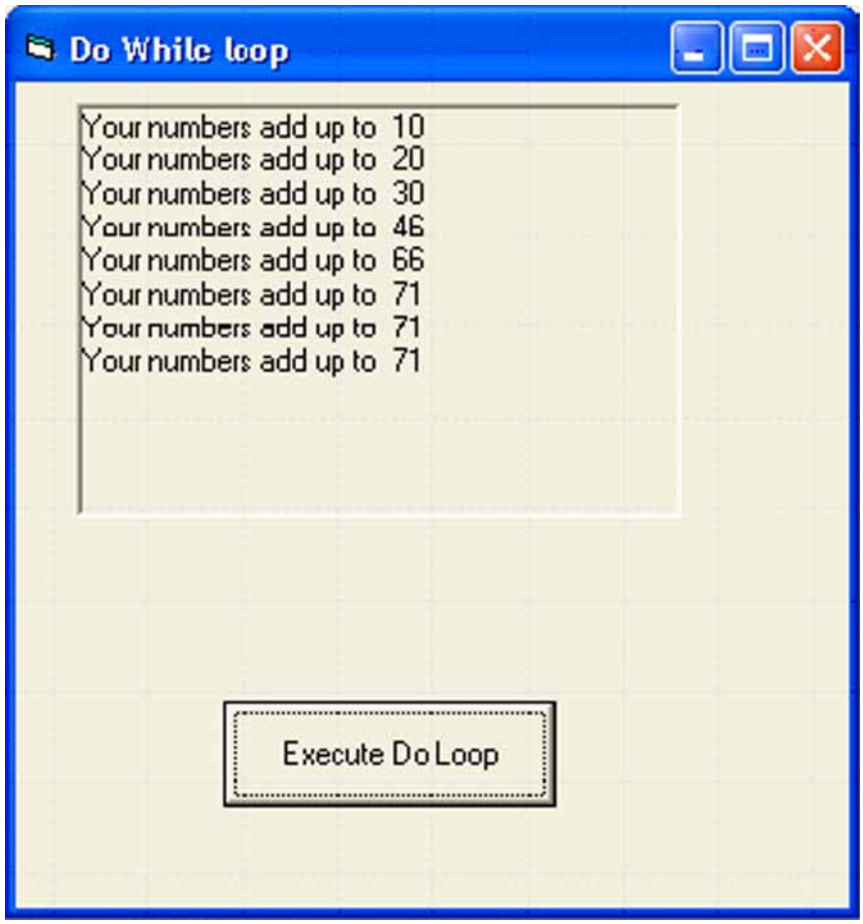


Figure 7.7:

Example 2 - Validating character input Problem:

A program is written to prompt the user to enter a character. The program continues to prompt until it receives a character other than a 'Y' or a 'y'.

Solution:

The algorithm is shown below:

```
1. display "Continue ?"
2. get input from the keyboard
3. do while user enters "Y" or user enters "y"
4.    display "Continue"
5.    get input from the keyboard
6. loop
7. display "Out of loop"
```

The Visual Basic program is shown in Code 7.6

```
Option Explicit

Private Sub Command1_Click()
  '17th February 2004
```

```
        'Program by Fred Fink

        'This program will illustrate the Do While..loop

        'Program CharEntry

        'This program prompts the user to enter a character.
        'It uses a WHILE loop to repeatedly check that a
        'character other than a 'Y' or a 'y' has been entered

        Dim Response As String

        PicDisplay.Print ("Do you want to continue ?")
        Response = InputBox("Y or y")
        Do While (Response = "Y") Or (Response = "y")
          Response = InputBox("Continue ?")
        Loop
        PicDisplay.Print ("Out of loop")

    End Sub
```

This file (Validate.txt), can be downloaded from the course web site.

Code 7.6

The program output is minimal; since it only responds to the correct input then the program will simply accept the character that is entered. Only when the incorrect character is entered will the program fail to enter the loop and come up with the message "Out of loop".

This code can be used as part of a larger program to validate input. If coded as a procedure then it can be called from within the main program. We will discuss procedures more fully later in the final topic.

Example - Range checking using a boolean variable

Problem: A program is written to ask the user to enter a value. If the value is outside the range expected the user will be prompted to enter another value. This is another example of validation, only in this program the use of a boolean variable is exemplified.

Solution:

The algorithm is shown below:

```
1. input "What size is needed?"
3. set value of ok to false
4. do while ok = false
5.    if (size >= low value) and (size <= high value) then
6.        set ok to true
7.        display "This size in range"
8.    else
9.        display "This size is out of range"
10.       display "Try entering another size"
```

```
11.       input "What size is needed?"
12.   end if
13. loop
14. display "Out of loop"
```

The Visual Basic program is shown in Code 7.7

```
Option Explicit

Private Sub Command1_Click()
  'program RangeCheck
  '18th February 2004
  'Program by Fred Fink

  'This program prompts the user to enter numeric value.
  'It uses a WHILE loop to repeatedly check that a
  'value which is between LowValue and HighValue

  Const LowValue As Integer = 10
  Const HighValue As Integer = 20

  Dim size As Integer
  Dim OK As Boolean

  OK = False
  size = InputBox("What size is required?")
  Do While Not OK
    If (size >= LowValue) And (size <= HighValue) Then
      OK = True
      Print size; " is in range"
       Else
      Print size; " is out of range, try again"
      size = InputBox("What size is required?")
    End If
  Loop
Print ("Out of loop")
End Sub
```

This file (RangeCheck.txt), can be downloaded from the course web site.

Code 7.7

Sample program output is seen in Figure 7.8

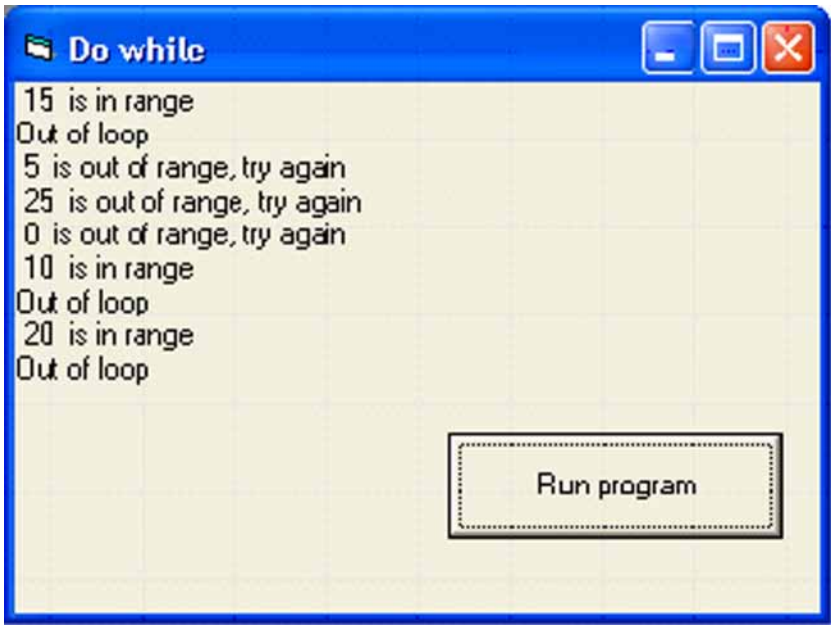


Figure 7.8:

Q11: What is wrong with the following?

```
n = 0  'assume n is an integer
do while n < 100
  value = n * n
loop
```

Q12: What is wrong with the following?

```
i = 1  'assume i is an integer
do while i <= 10
  Print(i)
i = i + 1
```

Q13: Write a `while` loop that calculates the sum of all numbers between 0 and 20 inclusive. Hint: use two integer variables, one for a loop counter and one for keeping a running total of the numbers

7.4.3 Do Until..Loop

The `while..do` loop performs the conditional test first and then executes the loop, so the statements within a loop may never be executed. The `do until` loop performs the statements first and then tests the condition. This means that the body of the loop is always executed at least once. This is the only difference, but a significant one between these looping constructs.

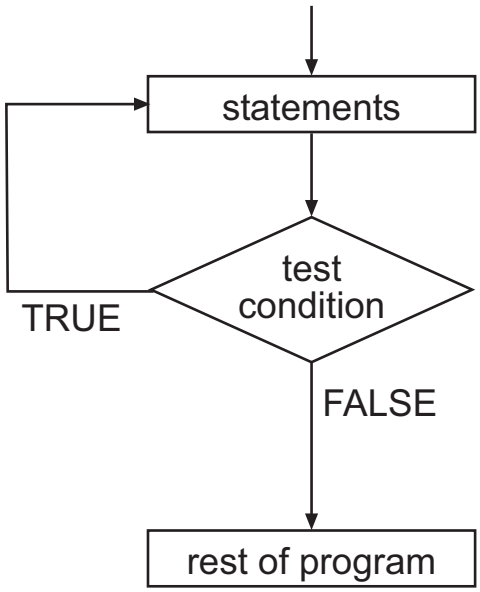


Figure 7.9:

The general form of a `do until.. loop` is:

```
Do Until test condition
    statements
loop
```

The statements in the body of the loop are executed repeatedly until the test condition is FALSE.

See Figure 7.9

The `do until..loop` is an example of a top tested *while false* structure. The variant `do loop..until` is an example of a bottom tested *while false* structure:

```
Do
   statements
Loop until test condition
```

The top tested loop can iterate between 0 and N times whereas the bottom tested loop may iterate between 1 and N. This means that using the bottom tested loop the iteration must occur at least once, so the condition can be tested immediately.

Some simple examples should clarify the situation.

Consider the simple lines of code representing a simple `do until..loop`

```
Dim Num As Integer

Do Until Num = 10
   Num = Num + 1
Loop

MsgBox Num
```

The equivalent `do while..loop` would be:

```
Dim Num As Integer

Do While Num < 10
   Num = Num + 1
Loop

MsgBox Num
```

Although both loops end up with similar results their methods are different. With the `do until loop` the condition `Num = 10` is set and the loop must be entered at least once to test this condition as `true`. `Num` is continually tested until the condition fails i.e. `Num` becomes $>$ 10 and the loop exits with `Num = 10`.

The equivalent `do..while` loop tests the condition in the first line and if `false` the loop will not be entered. If true then the loop will run until `Num = 10` when the loop exits.

When should you use one and when the other? If you know it is safe to run the code at least once, probably you should use the `do..until` loop. If you *must* run the code at least once then again a `do..until` loop is a good solution. If there is any reason to doubt the value of any variables etc. in the loop, then you should always use the `do..while` loop. Menus often use `do..until` loops as you know that the menu needs to be run at least once for the user to see it!

A number of worked solutions to problems are given here - make sure you understand what is going on in these examples. You may wish to try running these programs for yourself and to experiment with making changes to the code to ensure you fully understand what is happening.

Example 1: Guessing an age with `do..until` and nested `else..If` statement

Problem: A program is written to ask the user guess an age, the value of which is stored as a program constant. The program counts the number of tries needed to guess the correct age and declares whether the guesses are high or low.

Solution:

The algorithm is shown below:

```
1. set the number of guesses to 0
2. do
3.    request an age from the user
4.    if guess > age output message
5     elsif guess < age output message
6.    endif
6.    add 1 to the number of guesses
7. loop until guess = age
8. display "The number of times guessed"
```

The Visual Basic program is shown in Code 7.8

```
Option Explicit

Private Sub Command1_Click()

'program MyAge

'17th February 2004
'Program by Fred Fink

'This program will prompt the user to guess an age which is stored
'in the program as a constant.  It will display the number of
'attempts needed to guess the correct age

Const MyAge As Integer = 21 'again

Dim ThisGuess As Integer, guesses As Integer

  guesses = 0
  Do
    ThisGuess = InputBox("Guess my age")
    guesses = guesses + 1
    If ThisGuess < MyAge Then
       PicDisplay.Print ThisGuess; " is too low! "
       ElseIf ThisGuess > MyAge Then
       PicDisplay.Print ThisGuess; " is too high! "
    End If
  Loop Until ThisGuess = MyAge
    PicDisplay.Print "Ok,so its "; MyAge & " but it took you ",
guesses; " tries"
End Sub
```

This file (GuessAge.txt), can be downloaded from the course web site.

Code 7.8

You may wonder why the `elseif` statement is used, since by default if `ThisGuess < MyAge` is false then `ThisGuess` must be true within the `if..endif` statement. This is so but when the initial condition is met `MyAge = ThisGuess` an error occurs stating that the value 21 is too low followed by the correct statement as the loop exits.

The output is shown in Figure 7.10

Figure 7.10:

Example 2: Use of two do until loops to output a number series like Fibonacci numbers and to validate the input.

Problem: Write a program to output numbers belonging to the Fibonacci series up to a specified maximum.

Solution

Fibonacci was a famous Italian mathematician who identified the following series of numbers:

```
1, 1, 2, 3, 5, 8, 13.....
```

Successive terms of the series are calculated by adding the previous two numbers.

The following algorithm will produce the series, given the first two values as input.

```
1.  Do
2.    input number of terms to output
3.   loop until input is within range
4.  input two starting values
5.  do
5.    compute numbers
7.    display numbers
8.  loop until number of terms have been output
```

```
Option Explicit
Private Sub Command1_Click()
  'program Fibonacci

  '17th February 2004
  'Program by Fred Fink

  'This program will prompt the user to input the maximum value
  'of terms to be displayed. Two starting values
  'are also input.

  Dim NoOfTerms As Integer, First As Integer, Second As Integer
  Dim Third As Integer, Count As Integer
  PicDisplay.Cls
  PicDisplay2.Cls
  Do
    NoOfTerms = InputBox("How many terms to display?") 'Validation check
  Loop Until (NoOfTerms > 0) And (NoOfTerms <= 16)
  PicDisplay2.Print NoOfTerms
  First = InputBox("Enter first number")
  Second = InputBox("Enter second number")
  Third = 0
  Count = 0
  PicDisplay.Print First; Second;
  Do
    Third = First + Second
    First = Third
    Second = Third + Second
    PicDisplay.Print First; Second;
    Count = Count + 1
  Loop Until Count = (NoOfTerms - 2) \ 2
End Sub
```

This file (Fibonacci.txt), can be downloaded from the course web site.

Code 7.9

The program input is restricted to values > 0 and < 15 otherwise the `do until loop` will never terminate until a value within the range is input.

If you find difficulty in following the logic of the program then perform a paper exercise running through the values of each variable as the program executes. This is called a *dry run* and is best done by means of a *trace table*:

Instructions	First	Second	Third	Output
Starting Values	1	1		
First Loop	2	3	2	1, 1
Second Loop	5	8	5	2, 3
Third Loop	13	21	13	5, 8

The two starting values are 1, 1 which are assigned to variables `First` and `Second`. Variable `Third` then takes on the value of `First + Second` which is 2. First now takes on the value of `Third` to become 2. Finally the variable `Second` takes on the value of `Third + Second` to become 3. The values of `First` and `Second` are now displayed to give the output:

```
1, 1, 2, 3, 5, 8.......
```

Because the output is `First` and `Second` the value of `Count` is halved otherwise the output would be twice that required i.e.

```
Do Until Count = (NoOfTerms - 2) \ 2
```

Also the value of `NoOfTerms` is decreased by 2 to take into account the first two values which are output first and are not part of the loop.

Sample program output is shown in Figure 7.11

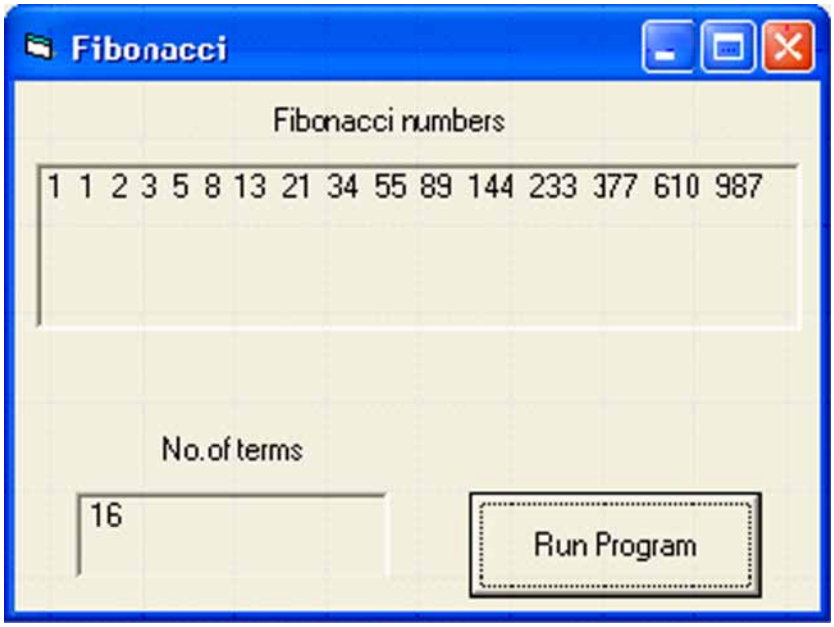


Figure 7.11:

7.4.4 Review Questions

Q14: Which one of the following statements is true regarding the **do while** loop?

a) The conditional statement is always satisfied
b) The block of code is entered first before the condition is tested
c) The loop need not be entered if the condition fails at the start
d) The loop terminates when the condition becomes true

Q15: Looping structures have several features in common. Which one of the following is NOT one of these features?

a) Selection
b) Increment
c) Initialisation
d) Condition

Q16: What is the output of the following program fragment that uses a **do while** loop?

```
i = 0
Do
   i = i + 1
   Print i;
Loop While i < 5
```

a) 0 1 2 3 4
b) 1 2 3 4 5
c) 0 1 2 3 4 5
d) 1 2 3 4 5 6

Q17: In the previous example the **do while** loop is replaced by a **do until** loop as follows:

```
i = 0
Do
   i = i + 1
   Print i;
Loop Until i < 5
```

a) 0
b) 1
c) 0 1 2 3 4 5
d) 1 2 3 4 5

Q18: What can be done to make the output of the program fragment in (3) the same as the program fragment in (2)?

a) Change last statement to Loop Until NOT i $<$ 5
b) Change last statement to Loop Until i $>$= 5
c) Use a for loop with counter variable i
d) Any one of the above

7.5 Arrays

In the exercises so far you have looked at simple data types, such as `single(real)` and `integer`. The next thing we want to look at is how related data items can be stored together using *arrays*. In this topic we will investigate:

1. the requirement for arrays
2. array declaration

Much of this material may be familiar to you already. However, it is necessary gain more practice in the use of arrays in order to better prepare you for a later topic on *standard algorithms* which makes extensive use of this type of data structure.

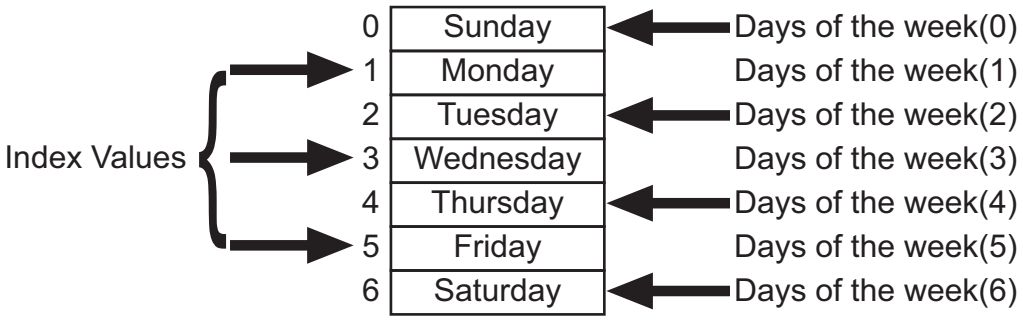
A data structure is a way of storing data in a computer in an organised way. One of the simplest data structures used in computer programming is called an array and is an example of a static data structure because it is of a fixed size within memory as defined within the structure of the program.

An array is a list of data items such that each item is uniquely identified by its position in the list. Also, an array is given a name, which is usually related to the group of data it holds. Some examples might be:

Data Items	**Array Name**
Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday	'Days of the week'
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	'Hexadecimal digits'
£1, £2, 50p, 20p, 10p, 5p, 2p, 1p	'Decimal coinage'

1-Dimensional array

Also called a Linear List the data items are simply stored in consecutive locations within a block of computer memory as follows:



Thus the data item '**Tuesday**' is stored in array location **Days of the week(2)** and the data item '**Saturday**' as `Days of the week(6)`. Each data item therefore is uniquely identified as an index or subscript.

Note that the index values are not actual memory addresses as such but simply refer to the data positions within the array.

Imagine you were to process the average temperature data in Edinburgh for the last 30 days. If each of the day's temperatures are stored as separate variables you

would require 30 distinct variables, and the storage and manipulation of these variables becomes difficult.

To overcome the above problem, you can define a variable called `temperature` which represents not a single value of temperature but an entire *set of temperatures*.

This is illustrated in Figure 7.12 showing an array structure called temperature and elements with the subscript ranging from 1 to 8.

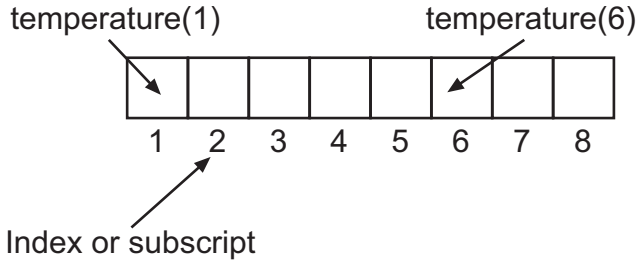


Figure 7.12: Representation of an array and indices

An individual array element can be used anywhere that a normal variable could be e.g.

```
temp = temperature(27)
average = (temperature(5) + temperature(6))/2.0
```

It should be obvious from the above that `temperature()` must be declared as a `real`. Can you say why ?

A value can be stored in an array simply by specifying the array element on the left hand side of the assignment operator (equals sign), e.g.

```
temperature(2) = 18.0
```

assigns the value 18.0 to be stored in element with subscript 2 of the `temperature` array.

```
temperature(7) = 14.5
```

assigns the value 14.5 to be stored in element index number 7 of the `temperature` array.

This is illustrated in Figure 7.13

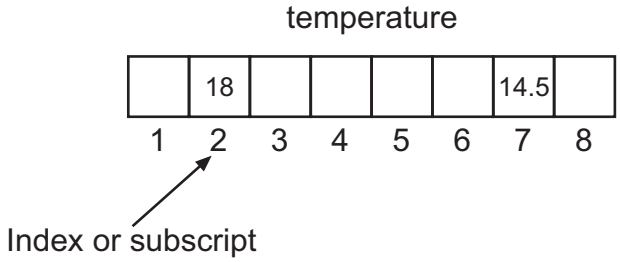


Figure 7.13: Representation of the temperature array with temperature(2) = 18.0 and temperature(7) = 14.5

More generally, an *integer* variable can be declared and used as the subscript, e.g.

```
temp = temperature(i)
```

which will take the value assigned in element number `i` of the array `temperature` and

assign it to the variable `temp`.

This means that if you want to access the elements of an array sequentially, you can do it using a `for..next` loop where the variable for the subscript is automatically incremented each time you go round the loop. Remember that the subscript variable can only be an `integer`. A `real` is not allowed. The examples within this topic will show you how you can access an array using a loop. This is the most common way of marching through an array from beginning to end, and as you will probably use this technique a lot in your programs.

7.5.1 Declaring arrays

Before you can use an array it must be declared at the beginning of the program with the other variable declarations. The declaration provides the name of the array, the number of elements and the data type of the elements.

Like variables arrays can be declared using the Visual Basic keywords `Dim`, `Public`, and `Private` that determines their scope. If `Dim` is used then the array is private to the procedure in which it is declared. `Public` makes the array visible from anywhere in the program, and `Private` (within the General section of a form or module) makes the array visible only to the form or module in which it's declared. but if you use Dim in the module's Declarations section, the array will be available to all procedures within the module.

Note that in Visual Basic, when an array is declared the first element of the array is usually 0 (zero) if not declared. It's possible, however, to force the first element of an array to be 1 by using the To statement.

All this will make sense with a few examples:

1. Declare an array called `MyName` to hold 6 integer values

```
Dim MyArray(5) As Integer
```

2. Declare a string array called `Names` to hold 3 string values

```
Dim Names(1 to 3) As String
```

3. Declare a public array called `Numbers` to hold 50 decimal values

```
Public Numbers(49) as single
```

Note: Visual Basic.NET does not support declarations using "to".

Errors

Note that you cannot mix array types. Once they have been declared as type then they cannot be used to store integer, real or string variables together. Arrays can be quite tricky and you will probably receive quite a few error messages that only occur at run time.

The most common error is to attempt to access an array element outwith the declared bounds e.g. accessing element `0` or `15` of an array with subscript range specified as `1..14`.

This type of error is so common that it has been given the name 'fencepost error'. It is

not a minor error as it can crash the system. If an array has an upper bound of `[9]` and you attempt to write something to item `[10]` then you could well be overwriting some variables which the computer has put in that location. You may be lucky and there is nothing important there. You may be unlucky. It does not matter whether your program works or not, it is flawed and incorrect. Sooner or later it will fail. Visual Basic traps this error and reports it to the user at run-time.

7.5.2 Initialising arrays

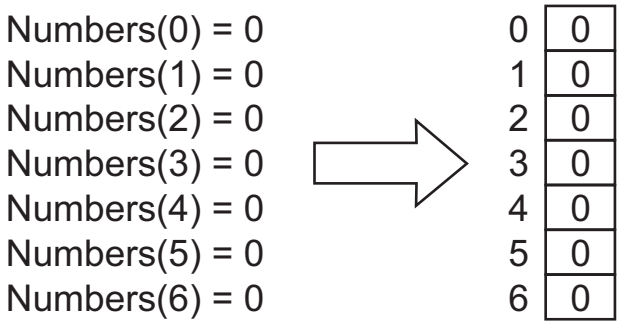
In computer programming arrays are used to manipulate data and the actions include:

- initialising the array
- reading data into the array
- searching the array for data items
- sorting data items

For this topic we will focus on the first three only.

Initialising an array simply means setting all the array elements to zero, null or a specified value.

Suppose we have an integer array called Numbers that can hold 7 values. To set all the locations to zero we can use the following statements:



To initialise an array of, say 1000 data elements this would be a very impractical way of doing it.

A much shorter method would be to use a loop structure as in the following piece of code:

```
Set Index to 0 {Initialising}
Do until Index = 6
  Numbers(Index)  =  0
  Index  =   Index   +   1
Loop
```

Initialisation is also useful to clear arrays of old data which otherwise might corrupt the new data being processed.

Whereas initialisation sets all array elements to be identical the same method can be used to input data. For example to read the days of the week into the array called Days:

```
Set Index  =  0  {Initialising}
Do Until Index = 6
  Input Days(Index)
  Index  =  Index  +  1
Loop
```

The following days would be input one by one into the array `Days`.

`"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"`

In this case when Index takes the value 0 the first data item - `Sunday` - will be read in to the array element `Days(1)`. Index then takes the value 1 and the process repeats itself until all the data has been entered.

7.5.3 Examples of using arrays

Example 1 - Using an integer variable to access an array

Problem: How can the array temperature be initialised to zero using a for loop?

Solution: A typical solution to this problem is shown here.

```
for count = 0 to 4
  temperature(count) = 0
next count
```

The first time round the loop the variable `count` has the value 0 and so the element with index number 0 of the array `temperature` is set to 0. The next time round the loop `count` has been incremented by 1 and has the value 1 and so element index number 1 of the array will be set to 0 and so on.

This one simple loop replaces 5 lines of code, i.e.

```
temperature(0) = 0
temperature(1) = 0
temperature(2) = 0
temperature(3) = 0
temperature(4) = 0
```

Example 2 - Writing out the contents of an array

Problem: How can the contents of an array be displayed using a `for` loop?

Solution: If you assume you have an array called Store which has 6 elements then you could use the following code to display the contents of each element.

```
for count = 0 to 5
  Print("element "; count; "= "; Store(count))
next count
```

For example, if the array Store held the values (3, 12, -4.6, 3.2, 0, -1), see Figure 7.14, then the information displayed would be:

Figure 7.14: Contents of a six-element array Store

This time one simple loop replaces 6 lines of code, i.e.

```
Print("element 0 = " Store(0))
Print("element 1 = " Store(1))
Print("element 2 = " Store(2))
Print("element 3 = " Store(3))
Print("element 4 = " Store(4))
Print("element 5 = " Store(5))
```

Look carefully, and you can see that the array can only be written out element by element.

You cannot use one statement to print the entire array. For example, it is wrong to say:

```
Print(Store)
```

and expect all elements of the array to be displayed.

A single element can be written or accessed at random - you are not forced to process the entire array to do this! Any one of the six lines of code above does the job of printing out that particular array element, so the code `Print('element 5 = ', Score(5))` outputs the *sixth* element of the array - the first element being element 0.

Example 3 - Reading user input into an array

Problem: How can data entered by the user at the keyboard be stored directly into the elements of an array?

Solution: Consider the situation where you want to store 4 user entered integers in an array called `mark`. You could use a `for next` loop to prompt and obtain input and to store the integers in the array, e.g.

```
for times = 0 to 3
    mark(times) = InputBox("Please input marks")
next times
```

For each loop the user will be asked to input a mark. After four marks have been entered the loop will terminate and the array mark will be storing the four value.

30 min

Using arrays

Learning Objective

Initialising, printing and putting data into arrays

Several examples using arrays have been illustrated in the course notes. Incorporate these program fragments into a working program which can do three things:

1. initialise each element of an array to zero;
2. allow you to input data in the form of floating point numbers directly into the array;
3. print out the contents of the array.

Try to make the program user-friendly by putting relevant information on the screen.

Finally, add a section to the above program which will print the array in reverse order after it has printed it conventionally.

20 min

Indexing arrays

Learning Objective

To understand how the index numbers are used in arrays.

An online array interaction is available to help you.

Answer the following questions. Use the arrays interaction on the course web site to help you find out the answers.

Q19: Begin with an array of 4 elements with a subscript range of 0..3. All values are initialised to 0, i.e. array = [0, 0, 0, 0]. Set array[1] = 3. What are the arrays contents now?

a) array = [0, 0, 0, 0]
b) array = [3, 0, 0, 0]
c) array = [0, 3, 0, 0]
d) array = [0, 0, 3, 0]
e) array = [0, 0, 0, 3]
f) none of the above

Q20: Using the same array set array[0] = 7. What are the array contents now?

a) array = [0, 0, 0, 0]
b) array = [3, 7, 0, 0]
c) array = [3, 0, 7, 0]
d) array = [7, 3, 0, 0]
e) array = [0, 3, 7, 0]
f) array = [7, 0, 3, 0]
g) array = [0, 7, 3, 0]
h) array = [0, 7, 0, 3]
i) array = [0, 0, 7, 3]

j) none of the above

Q21: Set up a 4 element array with a subscript range of 0..3 with the following values: array = [3, 6, 2, 8]. What is the value of array[2]?

a) 3
b) 6
c) 2
d) 8
e) none of the above

Q22: In the same array as the previous question, what is the value of the element with index 0?

a) 3
b) 6
c) 2
d) 8
e) none of the above

Q23: Set up an 8 element array with a subscript range of 0..7, e.g.. array = [8, 23, 5, 19, 3, 0, 7, 52]. Now set array[1] = 12. What are the array contents now?

a) array = [8, 23, 5, 19, 3, 0, 7, 52]
b) array = [12, 23, 5, 19, 3, 0, 7, 52]
c) array = [8, 12, 5, 19, 3, 0, 7, 52]
d) array = [8, 23, 12, 19, 3, 0, 7, 52]
e) array = [8, 23, 5, 12, 3, 0, 7, 52]
f) array = [8, 23, 5, 19, 12, 0, 7, 52]
g) array = [8, 23, 5, 19, 3, 12, 7, 52]
h) array = [8, 23, 5, 19, 3, 0, 12, 52]
i) array = [8, 23, 5, 19, 3, 0, 7, 12]
j) none of the above

Q24: Again, starting with an 8 element array with a subscript range of 0..7 e.g. array = [8, 23, 5, 19, 3, 0, 7, 52]. Set array[3] = 9, element with index 6 = 11 and array[6] = 4. What are the array contents now?

a) array = [8, 23, 5, 9, 3, 0, 4, 52]
b) array = [8, 23, 5, 19, 9, 4, 7, 52]
c) array = [8, 23, 5, 9, 3, 4, 7, 52]
d) array = [8, 23, 5, 19, 9, 4, 4, 52]
e) none of the above

Q25: Set up an array that can hold 4 values with a subscript range of 0..3. Initialise the array to hold all 0 values. Now set the following values:

- array[1] = 9
- array[2] = 3

What are the contents of the array now? (express in format of array = [3, 2, 5, 0]).

Q26: Set up an array that can hold 10 values with a subscript range of 0..9 Initialise the array to hold all 0 values. Now set the following values:

- array[4] = 6
- array[8] = 2
- array[1] = 7
- array[7] = 3

What are the contents of the array now? (express in format of array = [3, 2, 5, 0]).

Q27: Write a program that will set each element in an array, called `myarray`, of 10 elements to the value of its index. The contents of the array are then printed out in a vertical line.

7.5.4 Review Questions

Q28: Initialising a 1-dimensional array means:

a) Setting all locations of an integer array to 0 (zero)
b) Setting all locations of a string array to null
c) Setting all locations of a real array to pre-determined values
d) All of the above options

Q29: The contents of a string array `message()` contain the following characters in successive memory locations:

H A P P Y B I R T H D A Y

The programming structure required to produce the above message could best be achieved using: (choose one)

a) A case statement
b) A for loop
c) A while loop
d) An until loop

Q30: An array `numbers()` is declared with 6 elements and initialised to contain the following values:

0 3 0 2 5 9

During a program run the array elements are changed as follows:

```
numbers(2) = 6
numbers(5) = 7
numbers(1) = 4
```

The array contents are now:

a) 0 3 0 2 5 9
b) 4 3 0 2 5 9
c) 4 6 0 2 7 0
d) 0 4 6 2 5 7

Q31: An array `value_1( )` has been declared in Visual Basic to be of type single and to hold 5 values. Which one of the following statements would produce an error when the program is run?

a) value_1(0) = "Hello"
b) value_1(4) = 89.45
c) value_1(1) = 5.6E37
d) value_1(2) = 16

Q32: What is the least **positive** value of the variable `index` that will cause the following code to fail?

```
Dim array_1(19) As integer
Dim X As integer, index As integer
...
array_1(index) = array_1(index + index)
...
```

a) 9
b) 10
c) 11
d) 12

7.5.5 Worked example programs

A number of worked solutions to problems are given here - make sure you understand what is going on in these examples. Run these programs for yourself and experiment with making changes to the code to ensure you understand what is happening. It is important that you do this - programming is a skill which is learned by doing. Just reading the examples will not give you that skill.

Since some of the programs can be quite complex they are dealt with in meaningful chunks and more of this will be dealt with later.

Example 1 - Calculating the average temperature

Problem: Write a program to access each of the temperatures of a room over 14 days, which are to be held in an array. Calculate the average temperature during the period. No display of the results is required.

Solution: First try: A typical solution to this problem is shown in Code 7.10

```
Option Explicit
Private Sub Command1_Click()
  'program CalculateAverageTemp

  'program to calculate average temperature
  '20th February 2004
  'Program by Fred Fink

  Dim Store(14) As Single
  Dim total As Single, average As Single, Temp As Single
  Dim day As Integer
```

```
    total = 0.0   'initialise total

    'program now puts the values into the Store() array....

    For day = 0 To 13
      Temp = InputBox("Enter temperature of day ")
      PicValues.Print day
      Store(day) = Temp
      total = total + Store(day)
      PicArray.Print Store(day)
    Next day
    average = total / 14
    PicResult.Print Format(average, "standard")

End Sub
```

This file (AverageTemp.txt), can be downloaded from the course web site.

Code 7.10

Second refinement

The program in Code 7.10 does not yet deal with inputting values into the `temperature` array. One solution to this could be to hardcode the `temperature` array values into the program, although this does not give a lot of flexibility for the user as the values must be changed in the program and the program recompiled for every new set of values used. Another would be to allow the user to enter them, which is the method we will use here. There are other input methods which could be used, such as reading a file, or downloading the information from a remote monitor, but we will stick to the straightforward method at the moment.

```
for day = 0 to 13
    day = InputBox("Enter temperature of day ")
next day
```

Inserting this section of code into code 10 plus the output statements produces the following Code 7.11

```
Option Explicit
Private Sub Command1_Click()
  'program CalculateAverageTemp

  'program to calculate average temperature
  '20th February 2004
  'Program by Fred Fink

  Dim Store(13) As Single
  Dim total As Single, average As Single, Temp As Single
  Dim day As Integer
```

```
  total = 0.0   'initialise total

  'program now puts the values into the Store() array....

  For day = 0 To 13
    Temp = InputBox("Enter temperature of day ")
    PicValues.Print day
    Store(day) = Temp
    total = total + Store(day)
    PicArray.Print Store(day)
  Next day
  average = total / 14
  PicResult.Print Format(average, "standard")

End Sub
```

Code 7.11

As the temperature values are entered they are stored in the array `Store()`. When the loop terminates the value of `average` is displayed.

The program output is shown in Figure 7.15

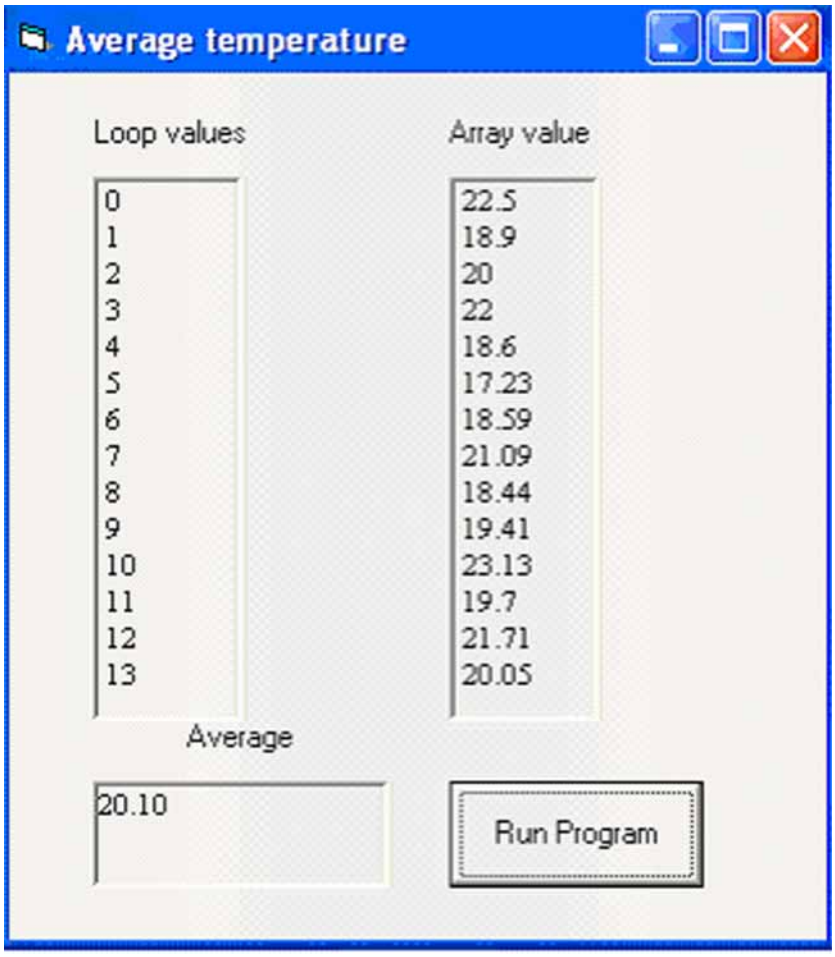


Figure 7.15:

Simulations

Using arrays makes it easy to **simulate** many numerical activities that otherwise would be fairly tedious to do manually. For example analysing the results of tossing a coin 1000 times, throwing die or spinning a roulette wheel can all be dealt with in one-dimensional arrays. Such simulations make use of random numbers that can be generated between upper and lower limits for example:

tossing a coin: random values 1 (for heads) or 2 (for tails)

throwing a die: random values between 1 and 6

roulette: random values between 0 and 49.

Random numbers

Visual Basic has a built-in function `RND()` that produces a random number from 0 to 1. The `randomise` function allows the `RND` function to start from a seed value and to produce a series of numbers based on the seed. Random numbers are generated internally using the computer's internal clock.

Since the random values produced are real, they must be converted into integers before being stored in the array. This can be achieved using another function `INT()`. Values are rounded down to the nearest integer.

For example:

```
Int(7.3)   =  7
Int(-6.8)  = -7
```

Example 2: Simulation of throwing a coin

Problem: Write a program to simulate the tossing of a coin up to1000 times determined by the user and output the number of heads and tails produced.

The program can be done in several sections:

1. Generate random numbers within the range
2. Store results of up to 1000 tosses
3. Display output

The following code segment will generate the random numbers and convert them to integers within the range 1 and 2:

```
Randomise
for toss = 0 to 999
    HeadTails(toss) = Int (rnd(2))
next toss
```

This will fill the array called `HeadsTails` with either 1's or 2s. `Rnd(2)` is used since the full statement is:

```
HeadsTails(toss) = Int((upper value - lower value) * rnd + lower)
```

Counting heads and tails

With 1000 numbers stored we can now scan the array and count the occurrences of 1's and 2s that represent heads and tails.

The following code segment should accomplish this:

```
For toss = 0 to 999
     If HeadsTails(toss) = 1 then
     heads = heads + 1
     else
     tails = tails + 1
     end if
next toss
```

Finally introduce the code to output the number of heads and tails.

What would you expect the two values to be ? Why ?

Code the program and run it a few times to see if you are correct.

The full Visual Basic program is shown in Code 7.12

```
Option Explicit
Dim HeadsTails(1 To 1000) As Integer
Private Sub Command1_Click()
  '20th February
  'Program by Fred Fink

  'This program simulates the tossing of a coin


  Dim toss As Integer, heads As Integer, tails As Integer
  Dim count As Integer

  Randomize
  PicHeads.Cls
  PicTails.Cls
  PicTosses.Cls
  count = InputBox("Enter number of tosses")

  For toss = 0 To count-1
    HeadsTails(toss) = Int((2 * Rnd) + 1) 'Generate numbers
  Next toss

  For toss = 0 To count-1                  'Count 1's and 2's
     If HeadsTails(toss) = 1 Then
     heads = heads + 1
     Else
     tails = tails + 1
     End If
  Next toss

  PicHeads.Print heads                    'Display results
  PicTails.Print tails
  PicTosses.Print count
End Sub
```

This file (HeadsTails.txt), can be downloaded from the course web site.

Code 7.12

Note that to make the program easier to follow two `for..next` loops are used but they could be combined into a single loop.

A typical output is shown in Figure 7.16

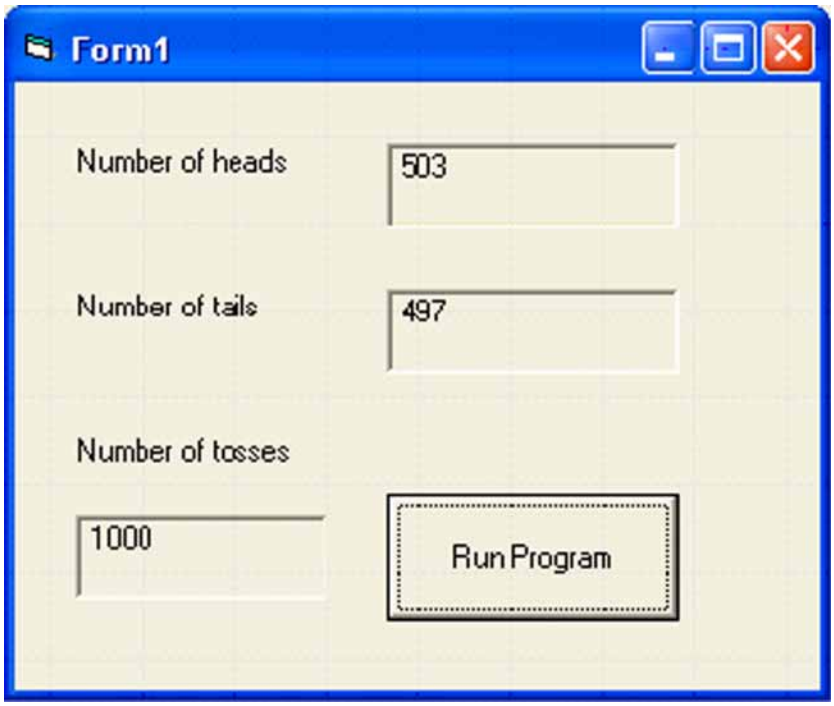


Figure 7.16:

The program can easily be modified to simulate throwing a die.

Exercise

Amend Code 7.12 to output the results of throwing a die about 50 times.

Example 3 -Testing for Palindromes using an array

Problem: Write a program which will read in a sequence of words. The characters will be read into an array and the program should then determine if the sequence is a palindrome.

Note: A palindrome is a sequence of number/characters/words etc. which is the same when read from either direction.

The following examples are palindromes, the sequence being the same when read from left to right or from right to left.

```
2 3 4 5 4 3 2

madam im adam

mas not a ton sam

a man a plan a canal panama

eve
```

This program may seem more complicated than it really is. If you follow the explanation you will see that by breaking it down into small steps, each of which can be written in Visual Basic you will chip away at the problem until it is done.

The solution to this problem assumes that you do not know how many words are going to be input.

Solution

The program does not know how big a sequence will be entered so an array of characters larger than required is declared. The program reads in the sequence of characters one at a time into the array, terminated by a full stop. It also checks that the array bounds are not breached by testing the condition:

```
count <= Size
```

using a `do..while` loop.

This is important, otherwise an array bounds error will likely occur. The constant `Size` is initially set to 30 but may be altered if large strings are used.

The condition in the `while..do` loop that checks whether "." was the last character entered is

```
palindromeChar <> "."   '<> means 'not equal to'
```

Once it has completed reading in the characters, you can determine the number of values read in. This is `count - 1`, not count since you are not interested in the last character. All it contains is the "." termination character which you do not want to use in the rest of the program.

Now you are ready to see whether the string is a palindrome. The array is searched through comparing the 1st element with the last, the 2nd element with the 2nd last etc. As soon as you find that they do not match you can terminate the search as you know that the sequence is not a palindrome. It is not necessary to continue searching the whole list of numbers as we know what we set out to find out.

Note that you only have to loop for `count <= length \ 2` as you compare the first half of the array to the second half. See following Figure 7.17:

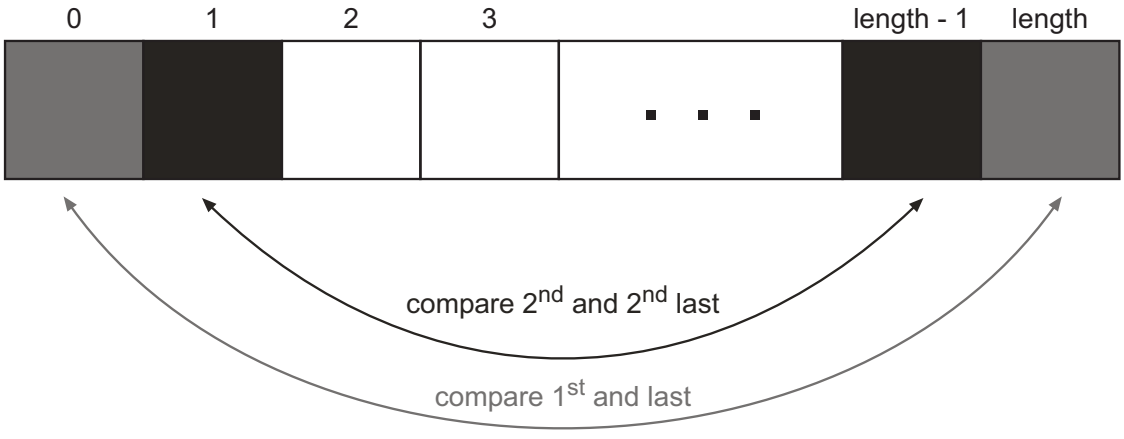


Figure 7.17: Comparing array elements for a palindrome

A possible solution is shown in Code 7.13

```
Private Sub Command1_Click()
  'program Palindrome
  '20th February 2004
  'Program by Fred Fink

  'This program will test an input string for a palindrome

  Const sizeArray As Integer = 30

  Dim Palarray(1 To sizeArray) As String
  Dim palindromeChar As String
  Dim length As Integer, count As Integer
  Dim isPalindrome As Boolean
  count = 1

  Do While (palindromeChar <> ".") And (count <= sizeArray)
     palindromeChar = InputBox("Enter your characters,one at a time.
Terminate by full stop")
     Palarray(count) = palindromeChar
     PicDisplay.Print palindromeChar;   'Output characters
     count = count + 1
  Loop

  length = count - 1        'to account for increment when "." is
                             read in

  'Now check for palindrome

   isPalindrome = True
   count = 1
   Do While (isPalindrome) And (count <= length \ 2)
      If Palarray(count) <> Palarray(length - count) Then
      isPalindrome = False
      Else
      count = count + 1
      End If
   Loop

   If isPalindrome Then
     PicDisplay.Print ("  is a palindrome")
     Else
     PicDisplay.Print ("  is not a palindrome")
  End If

End Sub
```

This file (Palindrome.txt), can be downloaded from the course web site.

Code 7.13

Examine the `do..while` statements carefully. You will see that the conditions are compound.

- the character read in must **not** be a full stop AND there must still be space left in the array
- if both these conditions are fulfilled, then the loop continues
- if either one or the other condition fails, then the loop terminates.

The `do while (isPalindrome) and (count <= length \ 2)` statement tests whether you have gone halfway through the array AND whether the boolean variable '`isPalindrome`' has been set to false.

This needs a bit more explanation:

- notice that we initialised `isPalindrome` to `true` at the beginning of the program
- if you look at the output statements right at the bottom of the code, you can see that if `isPalindrome` is still true, then the array is a palindrome
- if `isPalindrome` has been set to `false` at any point, then the array is not a palindrome
- this is what a boolean *flag* is for. You set it to a state - here it is either `true` or `false` - and you use this value to see whether a certain condition still holds true
- so what might set `isPalindrome` to `false`? It is the code inside the `while..do` loop which compares the first and last items
- if they are *not* the same, the boolean flag is set to `false`. Otherwise nothing happens
- then the second, and second from last are compared..... and so on.

In terms of loop tests this is probably about as complicated as it gets. If you do not understand it this time around, just be patient - you will probably find that you need to code something like this yourself in the future and you can come back to this example. By actually doing it, the difficulties seem much less than by reading it as you are doing now.

When you do code it successfully you will results like those shown in Figure 7.18

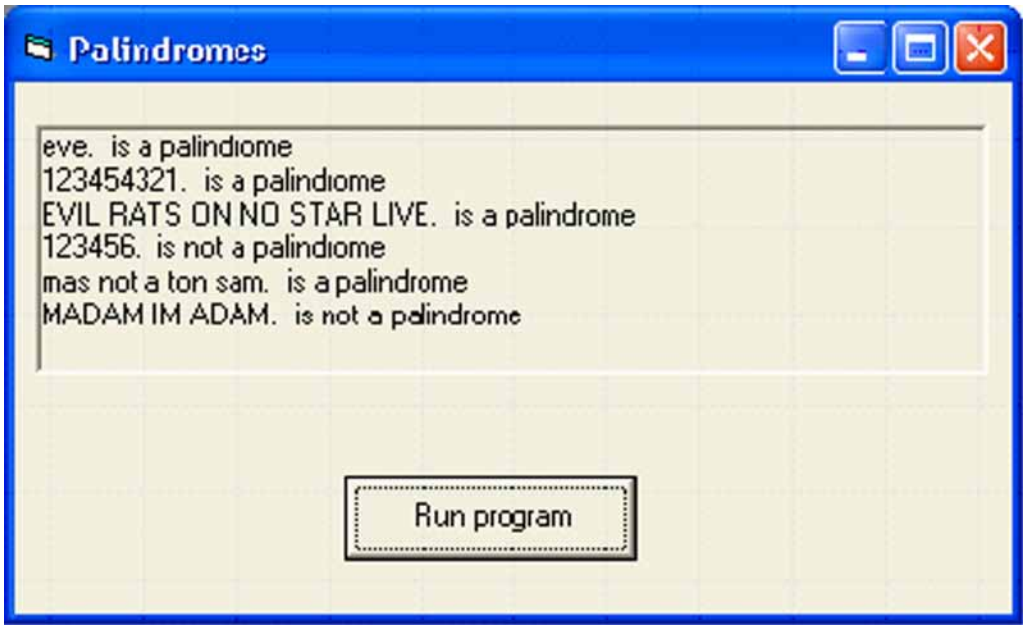


Figure 7.18:

There are other methods, probably much simpler, of writing programs to test for palindromes. However it is essential that you understand the use of arrays since they are important data structures in all fields of computing. You will see more in the use of arrays in the final topic.

Sentence completion - arrays

On the Web is a interactivity. You should now complete this task.

Comparing arrays and encoding the results

30 min

Declare three 20-element arrays, X, Y, Z. Write a program to read 20 integers into each of the two integer arrays `X` and `Y`. The program will prompt the user to enter the values into each of the arrays. The program will then compare each of the elements of `X` to the corresponding element in `Y`. Then, in the corresponding element of a third array `Z`, store the following values, Table 7.2.

Table 7.2

`Z` **Element Value**	**Condition**
1	if the element in `X` is larger than the element in `Y`
0	if the element in `X` is equal to the element in `Y`
-1	if the element in `X` is less than the element in `Y`

Then print out a three column table displaying the contents of the arrays `X`, `Y` and `Z`. Make up your own test data and write down in three columns the number you input for X, the number you input for Y and the result you got for Z

30 min

Comparing arrays - extension

If you found the previous exercise absurdly easy, then you can follow up with this one.

Extend the above program by adding two further columns to the output giving each line five columns. This is what you add: (column 4) a count of the number of elements of `X` that exceed `Y`, and (column 5) a count of the number of elements of `X` that are less than `Y`.

7.6 Summary

The following summary points are related to the learning objectives in the topic introduction:

- understand and be able to use the 'for ..next' structure;
- understand and be able to use the 'do..while' structure and variant;
- understand and be able to use the do..until structure and variant;
- how to declare 1-D arrays;
- initialise a 1-D array;
- manipulate data held in 1-D arrays.

7.7 End of topic test

An online assessment is provided to help you review this topic.

Topic 8

Procedures and Standard Algorithms

Contents

Prerequisite knowledge

Before studying this topic you should be able to:

- *describe and exemplify pre-defined functions.*
- *recognise appropriate use of the following standard algorithms:*
  - *input validation;*
  - *find min/max;*
  - *count occurrences;*
  - *linear search.*

Learning Objectives

- *understand and be able to use procedures in programs*
- *understand and be able to use functions and user-defined functions in programs*
- *understand how the use of procedures and functions aids modularity of programs*
- *understand parameters and how they are passed (in,out,in/out)*
- *understand parameter call by value and call by reference*
- *be able to describe in pseudocode and implement standard algorithms: linear search, counting occurrences and finding maximum/minimum*

Revision

Q1: A piece of programming code contains a validation routine. This is to ensure:

a) That the program produces the correct output
b) That the input data is within specified limits
c) That the output is within specified limits
d) That the input data is restricted to characters only

Q2: Programming languages usually contain a collection of pre-defined functions. Which one of the following statements is true?

a) Programming time can be saved
b) Functions produce a single value
c) Functions can be used with or without parameters
d) All of the above

Q3: A linear search is performed on the following data contained in a 1-D array:

Glasgow, Edinburgh, Falkirk, Perth, Inverness, Dumfries, St Andrews, Dundee

How many comparisons are made before the search item Dumfries is found?

a) 4
b) 5
c) 6
d) 7

Q4: What is meant by the term standard algorithm?

a) Universal code to solve all problems
b) A sequence of instructions that can be used to solve a particular problem
c) Code that makes a program more reliable
d) Any program written in a high level language

Q5: Of the following standard algorithms, which one is common to them all?

Linear search, counting occurrences, finding maximum/minimum

a) Linear search
b) Counting occurrences
c) Finding maximum/minimum
d) None of them

8.1 Introduction

This final topic introduces further modular programming concepts. Programs are built up from individual blocks of code that represent a procedure or function. The main program will then consist of a series of procedure/function calls and this, in itself does much to make any program more readable and easier to maintain.

The fairly complex issue of parameters and parameter passing is covered in detail and exemplified by working solutions.

Finally three common algorithms for Higher Grade Software Development Process are discussed with exemplar code for each supplied.

8.2 Modularity

You will recall from Topics 3 that an important aspect of the design phase was the idea of refining a complex program by progressively breaking it down into smaller, easier-to-solve units *(top-down design with step-wise refinement)*. Each of these units, called modules or program blocks, can be independently coded in a high level language and then amalgamated to form the complete program. The use of structure charts emphasise this modular design and also the relationship between all the modules in the software system.

Modularity was the basis of "structured programming", a concept which evolved in the 60s and 70s when procedural or imperative languages came to the fore. Today it is still an important aspect of modern programming techniques and even lends itself to current object-oriented programming.

In comparison to commercial software the programs that you will be writing in Visual Basic will be relatively short and you might not see the need to use modular techniques. The Windows operating system, for example contains well over 30 million lines of code. Just think of trying to program this in one large block of code! This would be an impossible task without the system being decomposed into many smaller, discrete units that can be easily modified and more readily debugged and compiled. Your own programs, therefore, however small they may be should embody the principles of modularity.

Visual Basic is an events driven programming language where the programming is done in a graphical environment. Users can click on various objects where each object is programmed independently to be able to response to user actions. A Visual Basic program is, therefore modular being made up of many subprograms, each having its own program code that can be executed independently. These subprograms are called procedures and functions.

Procedures and functions provide a means of producing structured programs, not only in Visual Basic but in other programming languages as well. The more complex a program becomes the more reason to employ strategies to make it easier to read and maintain. Procedures and functions help in this respect by breaking up what would be extensive lines of intricate code into more meaningful and logical sections.

As a consequence:

- the repetition of lines of code is avoided. Rather than repeating the same operations several different times in a program, the code can be placed in a procedure or function. The procedure is then called as many times as necessary without re-writing the code;
- it allows testing of the procedure code to take place in isolation from the main program. Each functional unit can be written independently by programming teams, thus making this part of the software development process more efficient;

- debugging of the main body of the program is simplified since each procedure can be individually tested;
- procedure code can be saved and re-used in future projects. **Module libraries**, for instance are repositories for useful chunks of code that can be accessed by programmers who could save time by using such code, instead of 're-inventing the wheel.

8.3 Procedures and Functions

Procedures and **functions** provide a means of producing structured programs, not only in Visual Basic but in other programming languages as well. The more complex a program becomes the more reason to employ strategies to make it easier to read and maintain. Procedures and functions help in this respect by breaking up what would be extensive lines of intricate code into more meaningful and logical sections.

As a consequence:

- the repetition of lines of code are avoided. Rather than repeating the same operations several different times in a program, the code can be placed in a procedure or function. The procedure is then called as many times as necessary without re-writing the code
- it allows testing of the procedure code to take place in isolation from the main program. Each functional unit can be written independently by programming teams, thus making this part of the software development process more efficient
- debugging of the main body of the program is simplified since each procedure can be individually tested
- procedure code can be saved and re-used in future projects. Module libraries, for instance are repositories for useful chunks of code that can be accessed by programmers who could save time by using such code, instead of 're-inventing the wheel.'

Procedures and functions are somewhat similar in their structure. They both consist of:

- a heading
- a declaration part (where necessary)
- an action part (a compound statement)

A procedure or function is activated by a call from the main program, after which the procedure or function executes its block of code and then terminates.

The flow of data between procedures, functions and the main program block is accomplished by the use of **parameters**, which will be discussed later.

8.3.1 Procedures

Visual Basic offers a variety of procedure types. The two that concern us here are:

- event procedures
- general procedures.

Up to now most of all the programming code you have seen has been made up of *event procedures* that activate sections of code when, say a command button is pressed. These procedures are named by Visual Basic by concatenating the name of the object code and the name of the event according to the syntax:

```
Sub Command_Click()
  End
End Sub
```

You will recognise this code which ends a Visual Basic program.

8.3.2 General Procedures

A general procedure, unlike an event procedure is user-defined with its block of code being called from the main program.

The basic structure for a procedure is:

```
Sub ProcedureName (formal parameters)
Declarations
Statements
End sub
```

The name, `ProcedureName` or identifier, is what is used when the procedure is called and the name conforms to the same rules of naming variables. Data is passed to and from the main program using the procedure `parameters` or `arguments` enclosed in parentheses. Not all procedures however need do have parameters. The *declarations* and *statements* follow an identical pattern to normal programming constructs.

Procedures are normally declared `private` and their scope is limited to other procedures and variables within the current form.

Here, to get us started is a simple little procedure that prints blank lines on a form. This might be useful for putting spaces into program output.

```
Private Sub BlankLines
   Print
   Print
   Print
End Sub
```

This could be called, in the action part of the main block, as:

```
Call BlankLines
```

Its use in a program could look like this:

```
   Private Sub command1_Click()
   'program lines
Print("Message 1.......")
Call BlankLines
Print("Message 2.......")
   End Sub

   Private Sub BlankLines
        Print
        Print
        Print
   End Sub
```

As it stands, `BlankLines` isn't very well structured. As programmers, we should always be a little wary when we find ourselves repeating code. It might be better to put the `Print` statements within a loop structure. The following refinement should produce a more acceptable program:

```
Private Sub BlankLines
 print("Message 1......")
 for n = 1 to 3
   Print
 Next n
 print("Message 2......")
End Sub
```

The program is still not very flexible. We should be able to tell the procedure how many blank lines to output.

This is achieved using parameters and we will come back to this section of code.

Exercise

A user is asked to input numerical values between 1 and 30. Write a section of code as a procedure to validate user input between these two values. Show how it would be called from the main program.

8.3.3 Parameter passing

Parameter passing is the main mechanism for transferring information between programs and sub procedures in Visual Basic. Parameters can be classified into various types, depending on how they deal with data values during a procedure call.

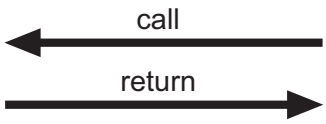
Other than using global variables, parameter passing using local variables is the way in which information is transmitted to and / or from called procedures in a program. The formal parameters can behave in one of three ways:

1. They can pass information to a sub program
2. They can receive information from a sub program
3. They can both send and receive information.

These three methods are described as:

1. IN mode or use of an IN parameter
2. OUT mode or use of an OUT parameter
3. IN/OUT mode or use of an IN/OUT parameter.

This is probably shown best in the following diagram:

Declared procedure Sub procedure(a, b, c) Formal parameters a, b, c	**Information flow**	**Calling procedure** procedure x, y, z Actual parameters x, y, z
a		x IN Mode
b		y OUT Mode
c		z IN/OUT Mode

In the diagram the declared procedure has three formal parameters `a`, `b` and `c` while the calling procedure has actual parameters `x`, `y` and `z`. During a procedure call the formal and actual parameters will interact as follows:

IN mode: information regarding `x` is passed to `a`

OUT mode: returning information from `b` to `y`

IN/OUT mode: information regarding `z` is passed to `c` where it is updated and returned to `z`

Make sure you understand the differences between the three modes.

The information that is passed can occur by two methods:

1. Call by value
2. Call by reference

8.3.4 Call by Value

Call by value is the most common parameter passing mechanism and is also the easiest to understand.

Recalling the section of code that outputs blank lines we can now introduce a **value parameter** to make the procedure much more flexible.

Here is the Visual Basic code (Code 8.1)

```
Private Sub Command_Click()
  'Modified program lines
  Print("Message 1.......")
  Call BlankLines(5)
  Print("Message 1.......")
End Sub

Private Sub BlankLines(ByVal Number as Integer)
  Dim count As Integer
  For count = 1 To Number)
      Print
  Next Count
End Sub
```

Code 8.1

By calling `BlankLines(5)` in the main program the formal parameter Number in the procedure declaration block is passed the actual parameter value 5. The procedure code is executed and then terminates.

Note that the formal parameter is preceded by the Visual Basic keyword `ByVal` to indicate a value parameter. This is optional but if left out then Visual Basic will default to `ByRef` which indicates a call by reference - see later.

The following worked example will help you understand passing by value:

Example 1: Use of procedures with value parameters

Problem: Write a program that outputs the square and square root of a number that is input by the user. This number will be passed to each of the procedures.

Solution

The following algorithm uses two procedures with *value* parameters:

```
1  Set up variables
2  Validate user input
3  call sub procedure "square"  'procedure 1
4  call sub procedure "sqrRoot" 'procedure 2
5  display results
6  end program
```

The full Visual Basic program is shown in Code 8.2.

```
Option Explicit

Private Sub Command1_Click()

   'Main Program

   'Program by Fred Fink
   '25th February 2004

   'This program exemplifies the use of value parameters

   Dim ActualValue As Integer

   Do
     ActualValue = InputBox("Input a value between 1 and 100")
   Loop Until (ActualValue >= 1) And (ActualValue <= 100)

   Call Square(ActualValue)              'Call procedure
   PicDisplay.Print
   Call SqrRoot(ActualValue)             'Call procedure

End Sub

Private Sub SqrRoot(ByVal number As Integer)   'Procedure declaration
   Dim Result As Single
   Result = Sqr(number)
   PicDisplay.Print "The square root of "; number; " is "; Format(Result,
                                                              "Fixed")
End Sub

Private Sub Square(ByVal number As Integer)    'Procedure declaration
   Dim Result As Integer
   Result = number ^ 2
   PicDisplay.Print "The square of "; number; " is "; Result
End Sub

Private Sub Command2_Click()
End
End Sub
```

This file (ValueParameter.txt), can be downloaded from the course web site.

Code 8.2

Examine this code carefully. Make sure you understand it's structure. The fixed format command is to output the square root to two decimal places.

Program output is shown in Figure 8.1

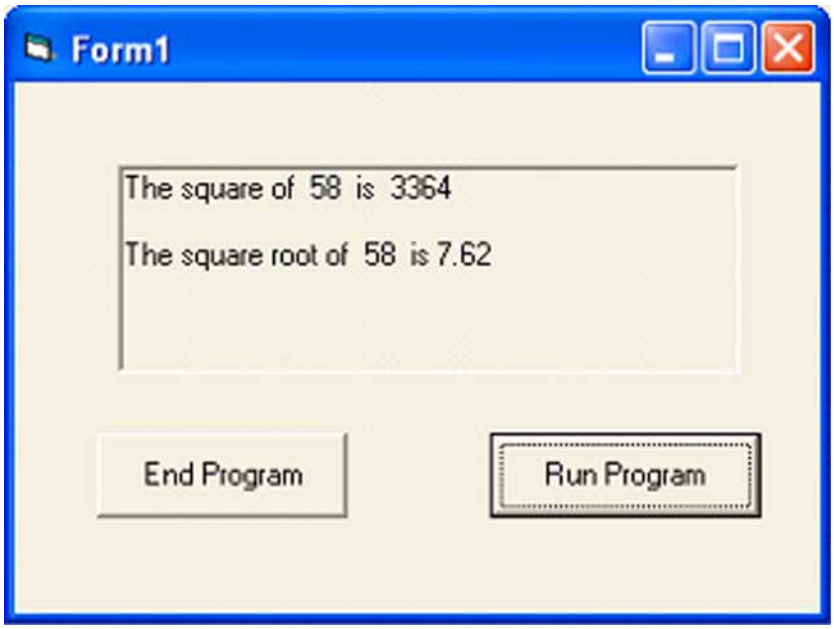
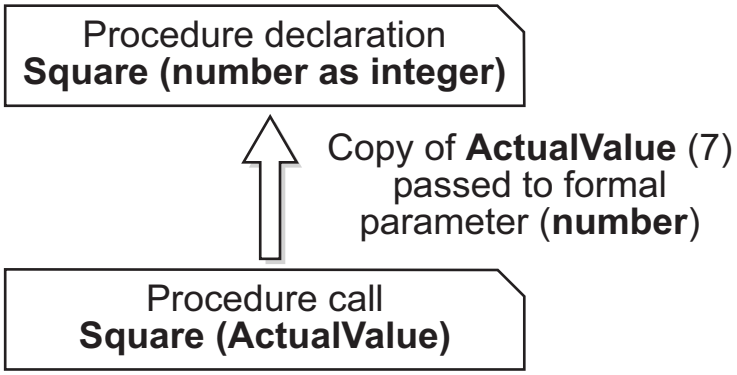


Figure 8.1:

In this program the formal parameter `number` was passed the value 7 in each of the two procedures. Consider the following diagram showing the procedure `Square`:



Once the value `7` has been sent to the procedure `Square` the code is run and the procedure then terminates.

Points to note:

1. The original value of the variable `ActualValue` in the main program remains unchanged. Any changes made by the procedure to the value passed are local to the procedure and not *passed back* to the main program. In other words the procedure can only modify a *copy* of the variable value and not the variable `ActualValue` itself. You will see more of this later.
2. the number of actual and formal parameters must be the same; in this there was only one.
3. each parameter must be the same data type for the procedure call to work.

In a procedure call the word `call` may be omitted with only the procedure name and parameters required. For example:

`Call Square(ActualValue)` becomes Square `ActualValue`

This notation will be used from now on in all programs.

8.3.5 Call by Reference

One of the disadvantages of call by value is that a copy of the actual parameter is always made in order to produce the formal parameter during the procedure call.

A **reference parameter** does not pass a value of a variable but instead passes the address of the variable.

Call by reference is used when information has to be passed out from a procedure to the main program. Here the variables inside the procedure body are allowed to reference the memory location of the actual variable that passed it. This means that as the actual parameters change as do the formal parameters.

It is important that you understand the difference between value and reference parameters.

Consider the following procedure codes where one uses call by value and the other call by reference:

```
Private Sub No_Change(ByVal Number As Integer)
  Number = Number + 10
End Sub


Private Sub Change(ByRef Number As Integer)
  Number = Number + 10
End Sub
```

If they are now executed within a program we can see the difference to the variable Test in the main program by calling each procedure.

The full Visual Basic program is shown in Code 8.3

```
Option Explicit

Private Sub Command1_Click()
  Dim Test As Integer
  Test = 10
  PicDisp1.Print Test
  Change Test
  No_Change Test
  PicDisp2.Print Test
End Sub

Private Sub No_Change(ByVal Number As Integer)
  Number = Number + 10
```

```
End Sub

Private Sub Change(ByRef Number As Integer)
  Number = Number + 10
End Sub
```

Code 8.3

Figure 8.2 shows the value of the variable Test before and after each call of the procedures.

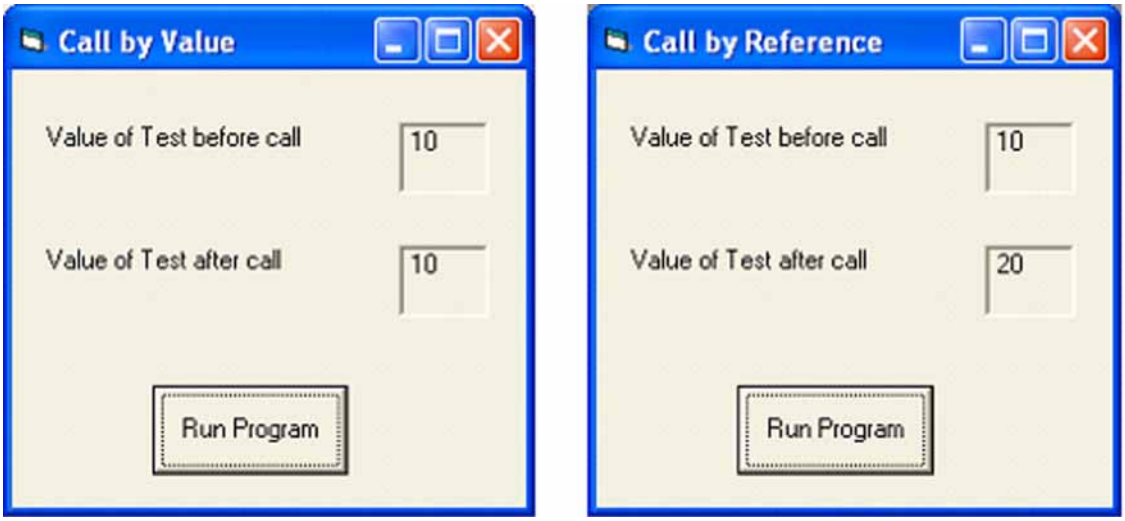
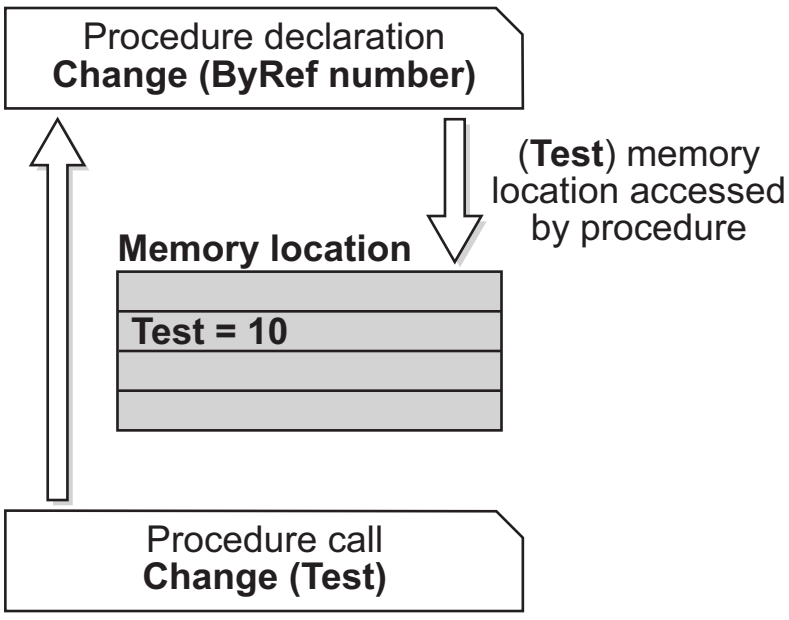


Figure 8.2:

Each procedure call produces the correct value of 20 for `Test` but only in call by reference is this value passed back to the main program. In call by value the main program knows nothing about the local changes made within the procedure.

The following diagram should explain what is happening:



The process of events is as follows:

1. The procedure call `Change(test)` is initiated;
2. The procedure declaration `Change` is accessed;
3. Procedure `Change` now accesses the memory location of variable `Test` and gets the numerical value 10. Formal parameter `number` now has the value 10.
4. Procedure `Change` code is activated and both `number` and `Test = 20`.

Points to note:

In this case a change is made to the program variable `Test` since it is the address of the variable that is referenced and not a copy. Since the formal parameter and the actual parameter have now been assigned the same memory address this means that any changes to the formal parameter are passed back to the main program.

It is important that you understand the differences between call by value and call by reference.

Don't worry if you have the material in this section difficult to follow first time round.

Remember!

- if a parameter is used only to transmit a *value* to a procedure then make it a value parameter
- if a parameter represents a result that is produced by the procedure to be used elsewhere in the program then make it a reference parameter.

Call by reference is the default condition for Visual Basic.

Note for Visual Basic.Net users the default condition is call by value.

Example 2: Program using reference parameters

Problem: A user enters two numbers at the keyboard. Write a program that swaps the positions of the numbers.

Solution

Consider the following procedural algorithm:

```
1   declare variables
2   GetNumbers(Number1, Number2)
3   BlankLines(2)
4   swap(Number1, Number2)
```

Notice that we are using the procedure `BlankLines`, the first procedure we met at the start.

The full Visual Basic program is shown in Code 8.4

```
Option Explicit

Private Sub Exchange_Click()
  Dim Number1 As Integer, Number2 As Integer

  GetNumbers Number1, Number2
  BlankLines(2)
  swap Number1, Number2

End Sub

Private Sub BlankLines(ByVal Num As Integer)
  Dim count As Integer
  For count = 1 To Num
    Display.Print
  Next count
End Sub

Private Sub swap(ByRef value1 As Integer, ByRef value2 As Integer)
  Dim temp As Integer
  Display.Print ("Numbers before swap = "); Tab(25); value1; " ";value2
  temp = value1
  value1 = value2
  value2 = temp
  BlankLines(1)
  Display.Print ("Numbers after swap = "); Tab(25); value1; " "; value2;
End Sub

Private Sub GetNumbers(ByRef Num1 As Integer, ByRef Num2 As Integer)
  Num1 = InputBox("Input first value")
  Num2 = InputBox("Input second value")
End Sub

Private Sub Command1_Click()
  End
End Sub
```

This file (RefParameter.txt), can be downloaded from the course web site.

Code 8.4

The first call of procedure `GetNumbers` produces the formal values for `Num1` and `Num2` through user input. These values are passed to the actual values `Number1` and `Number2`.

`BlankLines(2)` produces two clear lines before output of results

The values are now swapped:

```
temp = 20
value1 = 40
value2 = temp
```

Notice the procedure `BlankLines(1)` is nested within procedure `GetNumbers`.

Results are displayed via procedure `GetNumbers`.

Typical output is shown in Figure 8.3

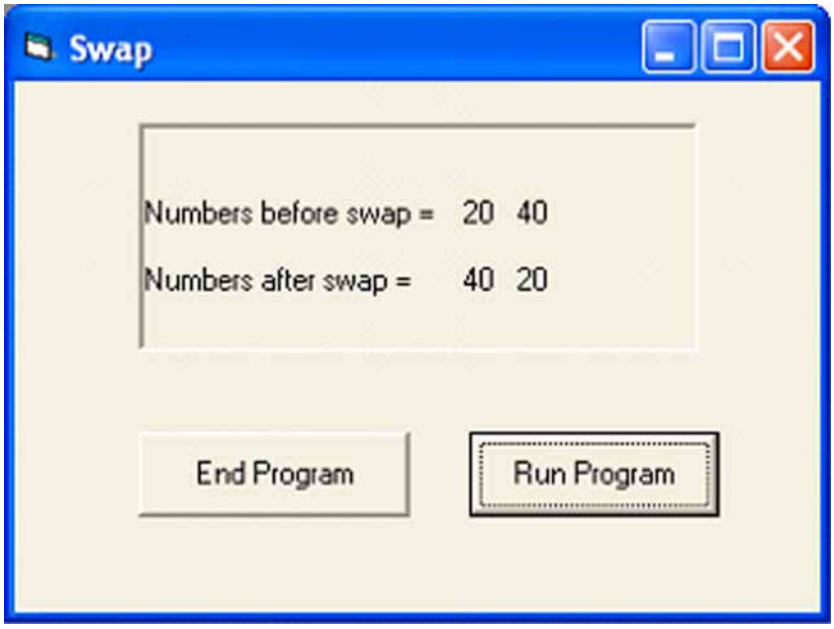


Figure 8.3:

8.3.6 Review Questions

Q6: The use of procedures in a program offers many advantages to the programmer. Which one of the following would represent such an advantage?

a) They must have meaningful names
b) Repetition of lines of code are avoided
c) Procedures can be nested within procedures
d) They can work with or without parameters

Q7: The structure of a procedure consists of three main parts. Which one of the following is not one of these parts?

a) A heading
b) A declaration
c) An action
d) A footer

Q8: Parameters can be classified as being IN, OUT and IN-OUT. Which one of the following statements is true regarding these parameters?

a) In IN parameter is read only
b) An OUT parameter is write only
c) An IN-OUT parameter is both read and write
d) All three options are true

Q9: In a procedure call by reference, which one of the following is true?

a) A reference parameter is an example of an IN parameter
b) Anything that happens to the formal parameter happens to the actual parameter
c) The memory address of the formal parameter is passed
d) All of the above

Q10: What would be the output of the following Visual Basic code?

```
Dim x As Integer, y As Integer
x = 5
y = 6
Print x, y;
result x, y
Print x, y

Sub result(ByRef num1 As Integer, num2 As Integer)
   num1 = num1 + 10
   num2 = num2 - 3
End Sub
```

a) 5 6 5 3
b) 5 6 3 5
c) 5 6 15 3
d) 5 6 3 15

8.4 Functions

A function is similar to a procedure, except it returns a value to the calling code. The basic structure for a function is:

```
Function functionName (parameters) As Data Type
    Declarations
    funcName = expression
End Function
```

Since a function returns a value it is used within expressions. The function name must, in the action part of the function, be assigned the value that is to be returned.

Some built-in Visual Basic functions have been used in many of the programming examples up to now. Table 8.1 shows some of the more common functions:

Table 8.1:

Name	Function	Example
ABS(X)	Returns the absolute value of X	ABS(5.6) = 5
INT(X)	Returns truncated integer part of X	INT(34.5) = 34
SQR(X)	Returns the square root of X	SQR(25) = 5
RND	Generates a random number between 0 and 1	X = (100*RND) will give random numbers from 1 to 100
TAB(X)	Outputs at print position 'X'	Print TAB(5)
SPC(X)	Outputs number of spaces between last printed position and the next	Print SPC(3)
ASC("X")	Returns the ASCII value of a character "X"	Print ASC("A") returns 65
CHR$(X)	Returns an ASCII value into a character	Print CHR$(68) returns "D"
VAL("X")	Returns the value of string "X"	VAL("76923") = 76923
STR$(X)	Returns the string of value X	STR$(1234) = "1234"

User-defined functions work in exactly the same way as Visual Basic in-built functions and extend the range of programming possibilities.

Here, as a simple example, is a little function that simply doubles the value sent to it.

```
function double (number as integer) as integer
 double = number * 2
end function
```

The function would then be called as

```
DoubleNumber = double(10)
```

We could combine this in a section of a program with a procedure to get a valid (positive) number:

```
'Program getNumbers
    Dim number integer
    getValidNumber (number)
    print ("Twice that is ", double (number))

  sub procedure getValidNumber (ByRef number as integer)
    do
       number = inputbox("Enter a number: "
    loop until (number > 0)
  end sub

  function double (number as integer)as integer
    double = number * 2
  end function
```

Exercise

Code the above section of code into a Visual Basic program and run it with different values.

Example 1: Use of a function

Problem: A program has to be written that uses a function to output the area of a circle, given the radius.

Solution

Consider the following algorithm:

```
1  declare variables
2  function area
3  area = pi * radius ^ 2
4  display output
```

The Visual Basic program is seen in Code 8.5

```
Option Explicit
Const Pi As Single = 3.141592
Private Sub Command1_Click()
  Dim Radius As Integer
  PicResult.Print Tab(4); "Radius"; Tab(16); "Area"
  PicResult.Print
  For Radius = 1 To 10
    PicResult.Print Tab(6); Radius; Tab(16); Format(Area(Radius),  "Fixed")
  Next Radius
End Sub
Function Area(Radius) As Single
  'This function will calculate area of circle
  Area = Pi * Radius ^ 2
End Function

Private Sub EndProgram_Click()
  End
End Sub
```

This file (FunctionArea.txt), can be downloaded from the course web site.

Code 8.5

The function call `Area(Radius)` is highlighted and the output is formatted to two decimal places. A loop structure is used to output the values between 1 and 10.

Rewrite this procedure as a function. It will return a valid integer. It will no longer need a parameter.

```
Sub getValidNumber (ByRef number As integer)
Dim numberOk As boolean
     do
           Print ("Input a number  ")
           numberOk = number >= 0
           if not numberOk then
              print ("The number must be greater than 0")
           end if
     loop until numberOk
End sub
```

Answer

```
function getValidNumber As integer
Dim numberOk As boolean
Dim number As Integer
     do
           print ("Input a number ")
           numberOk = number >= 0
           if not numberOk then
              print ("The number must be greater than 0")
           end if
     loop until numberOk
     getValidNumber = number
end function
```

Write a program to add two validated numbers together. First use the sum function and the `getValidNumber` procedure. Then write another program using the sum and `getValidNumber` functions.

Answer

Using the procedures:

```
'program sum1
Dim first As integer, second As integer

Sub getValidNumber (ByRef number As integer)
Dim numberOk As boolean
     do
           print ("Input a number ")
           numberOk = number > 0;
           if not numberOk then
              print("Make it positive ")
           end if
     loop until numberOk
end sub

function sum (a as integer, b as integer)as integer
     sum = a + b
```

```
end function

'Main program
     getValidNumber (first)
     getValidNumber (second)
     print("The total is ", sum (first, second))
```

Using the functions:

```
'program sum2;
Dim first as integer, second as integer

function getValidNumber as integer
Dim numberOk as boolean
Dim number as integer
     do
           print ("Input a number ")
           numberOk = number > 0
           if not numberOk then
              print("Make it positive ")
     loop until numberOk
     getValidNumber = number
end function

function sum (a as integer, b as integer)as integer
     sum = a + b
end function

'Main program
     first = getValidNumber
     second = getValidNumber
     print ("The total is ", sum (first, second))
```

Turning a procedure into a function

Rewrite this procedure as a function. It will return a valid integer. It will no longer need a parameter.

```
sub getValidNumber(ByRef number as integer)
Dim numberOK as Boolean
   do
     number = Inputbox("Input a number")
     numberOK = number > 0
     if not numberOK then
     Print("The number must be greater than 0")
     end if
   loop until numberOK
End sub
```

Using procedures or functions

Write a program to add two validated numbers together. First use the sum function and the getValidNumber procedure. Then write another program using the sum and getValidNumber functions.

Functions

On the Web is a interactivity. You should now complete this task.

8.5 Recursion

Recursion is a powerful programming technique since it can use the minimum of code to achieve maximum output. It should be used with care however since the action is an avid user of computer memory space.

Procedures and functions can be called from other procedures and functions - they are said to be nested - but calling a procedure or function within itself may seem strange. This is recursion and the procedure or function is called recursive.

Here is a simple example to print out the values from 1 to 10.

```
Option Explicit

Private Sub Command1_Click()
'25th February 2004
'Program by Fred Fink
'Program to illustrate recursion

Counter (10)  'procedure call

End Sub

Private Sub Counter(ByVal digit As Integer)
If digit > 0 Then
Counter (digit - 1)  'recursive call
PicDisplay.Print digit
End If
End Sub

Private Sub Command2_Click()
End
End Sub
```

The procedure Counter calls itself with the value digit -1 which is 10.

```
Counter (digit - 1)
```

This is repeated until the value of `digit` is reduced to 1. The clever bit now starts. The value of `digit` is output as `1` followed by the result of the procedure's previous call which

was 2. This continues until the final call is output as 10. The first procedure call is the last to displayed!

For each activation of the procedure a new set of local loop variables and value parameters are created and are stored in a dynamic memory structure call a stack. A stack is referred to as a LIFO (Last In First Out Structure) and can be created by the system for dealing with recursion and other arithmetical processes.

The educational language LOGO depends very heavily on recursion for its intuitive graphics displays.

Figure 8.4 shows the contents of the stack at the end of the procedure calls and Figure 8.5 shows program output.

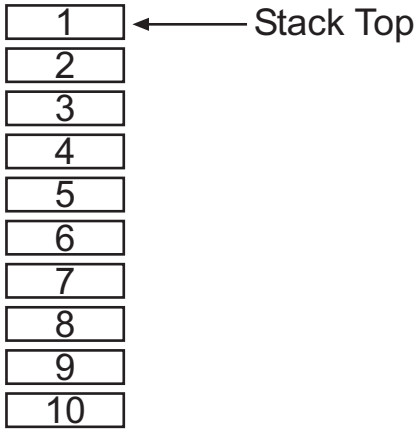


Figure 8.4:

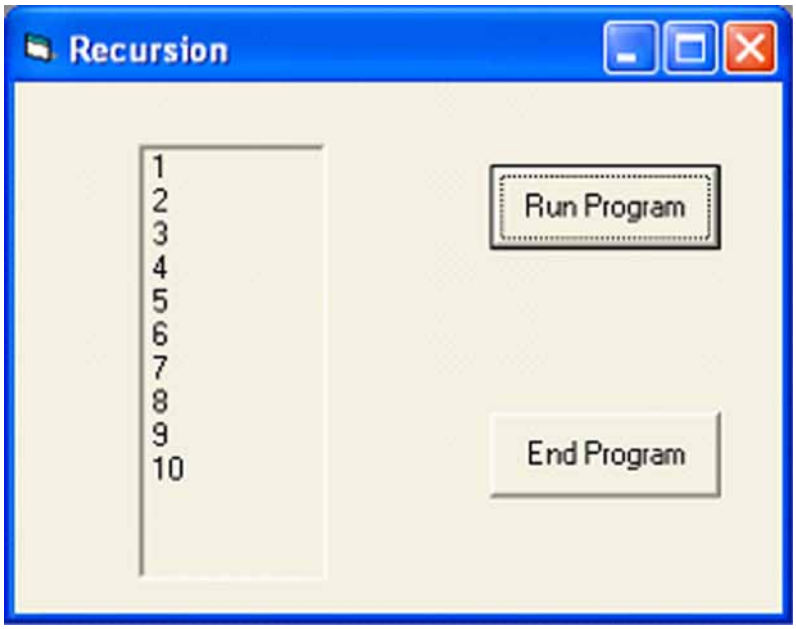


Figure 8.5:

Exercise

How could you amend the program to print the values out in reverse order?

Answer

Interchange the `If` and `Print` statements

Example 2

Recursion cannot be covered without a look at the factorial function. A factorial is a number multiplied by every other integer below itself, down to 1. For example, the factorial of the number 5 is:

```
Factorial 5 =  5 * 4 * 3 * 2 * 1
```

This give `factorial (5)` the value 120.

From this example it can be seen that `factorial (5) = 5 * factorial(4)`.

Similarly `factorial(4) = 4 * factorial(3)` and so on down to 1.

A general rule for calculating factorial numbers is:

```
factorial(n) = n * factorial(n-1)
```

The above rule terminates when n = 1, as the factorial of 1 is 1.

The following example in Code 8.6 calculates the Factorial of a number using recursion:

```
Option Explicit

Private Sub Command1_Click()
  Dim Result As Double, Number As Double
  For Number = 1 To 7
     Result = factorial(Number)
     Print "Factorial "; Number; " = "; Result
  Next Number
End Sub

Private Function factorial(ByVal Num As Double) As Double
    If Num = 1 Then        Śfactorial of 1 is 1
        factorial = 1
    Else
        factorial = Num * factorial(Num - 1)  'recursive call
    End If
End Function

Private Sub Command2_Click()
End
End Sub
```

This file (Factorial.txt), can be downloaded from the course web site.

Code 8.6

From the local value of `Number` in the `for..next` loop the first call of

```
factorial = Num * factorial(Num - 1)
```

takes on the value 1 which is displayed. Next the value 2 is passed and the result 2 * 1 displayed. This continues until the loop value ends at 7 when factorial(7) will be displayed.

Exercise

The previous program may be coded without recursion using a looping structure. See if this alternative version can produce factorials of numbers higher than the recursion method.

Hint: An other general rule for calculating factorials is:

```
factorial(n)  = n * (n - 1) * (n - 2).....1
```

Base your program on the following algorithm:

```
1    declare variables
2    do
3       input a value to get factorial
4    loop until n > 0
5    factorial = 1
6    do
7       factorial = factorial *  n
8       n = n - 1
9    loop until n = 0
10  display factorial
```

Recursive procedures

Recursive procedures are often used in programming, for example in sorting arrays or in searching through them.

A famous example of a recursive procedure concerns the problem of the **Towers of Hanoi**. This was a game or toy that came out in Victorian times, consisting of three upright rods and a set of discs of graduated size that went on the rods. The story that went with it was that, in a temple in Hanoi, the monks performed part of their devotions by moving similar discs of bronze from one rod to another, never moving a disc so that it lay on top of a smaller one. There were 72 discs. When the monks had finally moved all the discs from one rod to another, the world would come to an end.

Our task is to write a program to spell out the moves that need to be made. You might like to consider for a moment how you would set about the task using iteration.

We have three rods, called `left`, `right` and `centre`. Our task is to move all the disks from the `left` to the `right`, using the `centre`.

If we look at the matter recursively, we could break down the problem into three steps. First we move all the discs but the largest from `left` to `centre`, using the `right` rod to help. Then we move the largest from the `left` to the `right`. Then we move the pile of smaller discs back on top of the largest: from the `centre` onto the `right`, using the `left` rod to help. That's the general case. The base case is when there's only one disc to move. And that's it: problem solved.

Note that the number of discs involved or whether that number is odd or even have no effect on the method of solution. If you made much of an attempt to solve the problem by iteration, you should be able to see that the recursive solution is at a higher level: we are thinking of what the solution should be and not bothering too much about how exactly it will be implemented. We trust to our analysis and to the powers of the recursive process to look after all the details.

Here's a program that works in this way. It's set up to move a stack of only four discs. Note that, within the machine's limits, the `moveTower` procedure would work without alteration for any number of discs. Note also that there's only one `if` statement involved. You might like to use strings instead of the `integer` parameters `left`, `right`, and `centre`. In terms of numbers, the program is moving discs from rod 1 to rod 2 using rod 3.

```
Option Explicit
'Program by Fred Fink
'1st March 2004
'This program emulates the Tower of Hanoi
'using recursive procedure calls.

Private Sub Command1_Click()
  moveTower 4, 1, 2, 3
End Sub

Sub moveDisc(ByVal left As Integer, ByVal right As Integer)
  PicOutput.Print ("Move a disc from "); left; " to"; right
End Sub

Sub moveTower(ByVal discs As Integer, ByVal left As Integer,
ByVal right As Integer, ByVal centre As Integer)
  If discs = 1 Then
    moveDisc left, right
  Else
  moveTower discs - 1, left, centre, right   'recursive call
  moveDisc left, right
  moveTower discs - 1, centre, right, left   'recursive call
  End If
End Sub

Private Sub Command2_Click()
  End
End Sub
```

This file (Hanoi.txt), can be downloaded from the course web site.

Typical output is shown in Figure 8.6

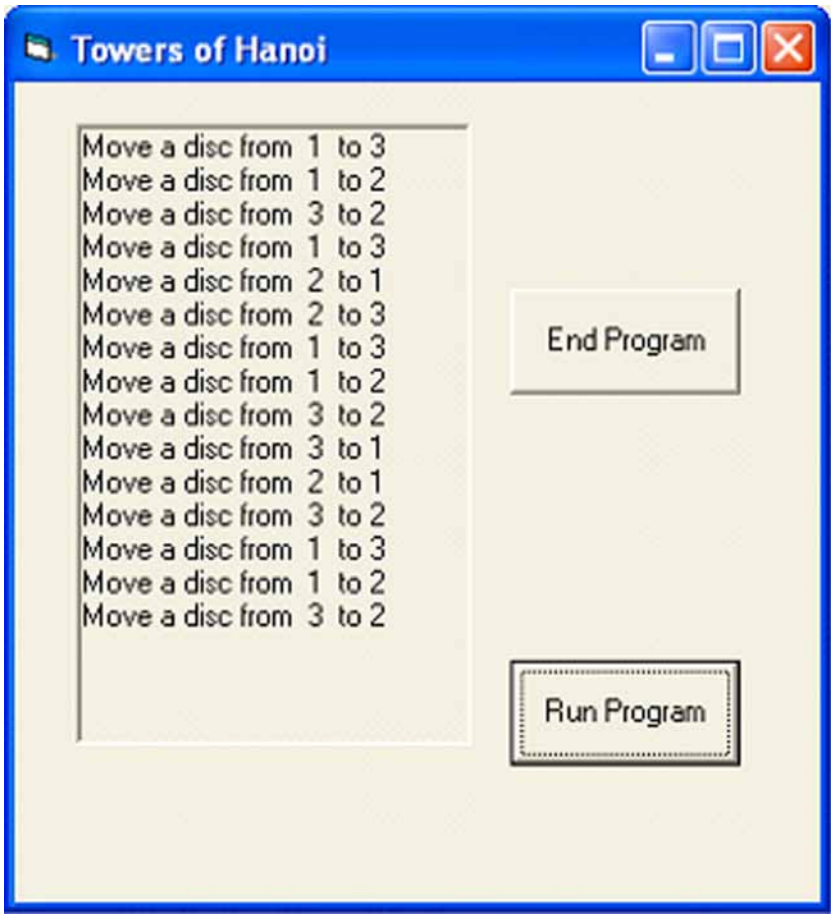


Figure 8.6:

Towers of Hanoi

There is a interactivity of the Towers of Hanoi online at this point.

8.5.1 Review Questions

Q11: Which one of the following statements regarding a function is false?

a) A function does not need to be declared
b) A function does not have parameters
c) A function returns a single value
d) A function can not call itself

Q12: Below are four functions. Which one does not belong in Visual Basic?

a) Inputbox
b) Val
c) Square
d) Rnd

Q13: Look at the Visual Basic statements below that use built-in functions. Which one represents a valid function call?

a) SQR(x) = x
b) y = ABS(-5)
c) 65 = CHR$("A")
d) z = SQR("z")

Q14: Procedures and functions can be recursive. This means that (choose one):

a) Procedures and functions can be nested
b) They are called twice within a program
c) The process can go on for ever
d) A procedure or function can call itself

Q15: What would be the value of `x` in the following expression?

```
x = 0.5 * factorial( 4)
```

a) 2
b) 5
c) 10
d) 12

8.6 Standard Algorithms

An algorithm is a finite sequence of steps which, when followed, will accomplish a particular task.

The term algorithm derives from the name of the mathematician, Mohammed ibn-Musa al-Khwarizmi (c.825AD) who was a mathematician and part of the royal court in Baghdad. Al-Khwarizmi's work is the likely source for the word *algebra* as well.

Figure 8.7: Cunieform tablet

There is ample proof that the use of 'algorithms' was evident around 2000 - 3000 BC by the ancient Sumerians whose work was inscribed on clay tablets using a cunieform cipher.

Translating this text revealed mathematical rules and astronomical data written in sexagesimal notation (base 60) from which remains 360 degrees for circular measure,

60 minutes per hour, 60 seconds per minute and so on.

A computer program can be viewed as an elaborate algorithm. In mathematics and computer science, an algorithm usually means a small procedure that solves a recurrent problem.

One major objective of this topic is to introduce you to common algorithms that have been tried and tested by programmers with knowledge of good algorithm design.

Many algorithms appear over and over again, in program after program. These are called **standard algorithms** or *common* algorithms.

Think about a word processing package which uses an algorithm to find all occurrences of a particular word in a block of text, or a spreadsheet package which uses an algorithm to find the maximum value in a range of cells. These packages make use of standard algorithms and it is worthwhile for every programmer to know them. When implemented these standard algorithms may become key components in a module library.

This section introduces 3 common algorithms used by programmers. Namely:

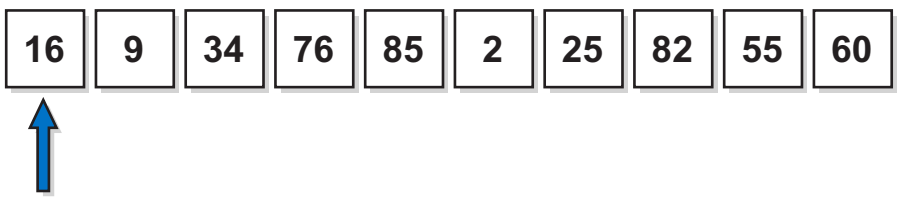
1. linear search
2. counting occurrences
3. finding maxima and minima.

It would be advantageous for you to have prior understanding of arrays, as examples given rely upon knowledge of accessing and manipulating such structures.

8.6.1 Linear Search

One task which computers frequently perform is to search through lists. For example, looking up a telephone number in a database, finding and replacing a word in a text file or looking for a particular stock item in a warehouse etc. The simplest way of doing this is to perform a linear search also called a sequential search, where the search begins at the first item in a list and continues searching through each item of the list in turn.

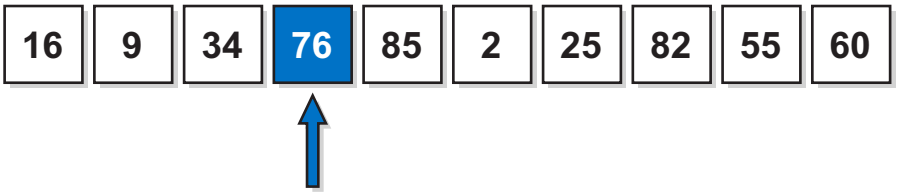
Linear search is the simplest search method to implement and understand. Starting with an array holding say, 10 numbers with a pointer indicating the first item the user inputs a search key. Scanning then takes place from left to right until the search key is found, if it exists in the list. Look at the list below:



Suppose the search key is 76.

1. 16 is compared to 76. Not the key so pointer moves on one place
2. 9 is compared to the key. Not equal so pointer moves on

3. 34 is compared to key. Not equal so pointer moves on
4. 76 compared with key. Success! Key found at position 4 in the list.



One method is to search the entire list from start to finish and stop only when the end of the list is reached. This algorithm is shown below:

Linear search 1 - searching the entire list

```
1.  prompt the user for the search value
2.  start at the first element in the list
3.  do while not end of list
4.    if the current element <> search value then
5.      move onto the next element in the list
6     else output search value and position
7     end if
8.  loop
```

This is not a very efficient way of doing things. Why bother to search the remainder of a list when the item has been found? There are instances when you will want to search the entire list e.g. if you wish to replace all words in a list that begin with 'L' to words that begin with 'P' then the search would stop when you reach the end of the list. The *counting occurrences algorithm* that you will meet next will accomplish this scenario.

The following algorithm makes use of a complex condition and a boolean variable:

Linear search 2 - stopping when the search value is found

```
1.  set found to false
2.  prompt the user for the search value
3.  start at the first element in the list
4.  do while (not end of list) and (found is false
5.      if the current element = search value then
6.        set found to true
7.        display message
8.        else
9.        move onto the next element in the list
10.      end if
11.  loop
12.  if not found then
13.    display message
14.  end if
```

The full Visual Basic program is shown in Code 8.7. Examine it carefully to see how it operates, based on the algorithm. The output of this can be seen in Figure 8.8

```
Option Explicit
Dim Found As Boolean
Dim SearchKey As Integer, Pointer As Integer, Fill As Integer
Dim List(15) As Variant
Private Sub Command1_Click()

'Program Linear_Search
'29th February 2004
'Program by Fred Fink

'This program will perform a linear search on
'an array holding 16 random integers between
'the values 1 and 99

Setup                                          'Call procedures
Populate_List List()
Enter_search_item SearchKey
Search_for_item Pointer, SearchKey, List()

End Sub


Private Sub Setup()                          'Initialise variables
  Randomize                                    'and clear output boxes
  Found = False
  PicList.Cls
  PicResult.Cls
  PicResult.Print
  Pointer = 0
End Sub

Private Sub Populate_List(ByRef List())        'Fill array with
  Dim Fill As Integer                                    '16 random numbers
  For Fill = 0 To 15
    List(Fill) = Int(99 * Rnd) + 1
    PicList.Print List(Fill);
  Next Fill
End Sub

Private Sub Enter_search_item(ByRef SearchKey) 'User input search item
  SearchKey = InputBox("Input search key")
End Sub

Private Sub Search_for_item(ByVal Pointer, ByVal SearchKey, ByRef List())
  Do While (Pointer <= 15) And (Not Found)

         If List(Pointer) = SearchKey Then
            Found = True
            PicResult.Print "Search item "; SearchKey; " found at position
```

```
                                                                    "; Pointer
            Else
              Pointer = Pointer + 1
            End If
      Loop

      If (Not Found) Then
        PicResult.Print "Search item"; SearchKey; "is not in list!"
      End If
    End Sub

    Private Sub Command2_Click()
      End
    End Sub
```

This file (Linear.txt), can be downloaded from the course web site.

Code 8.7

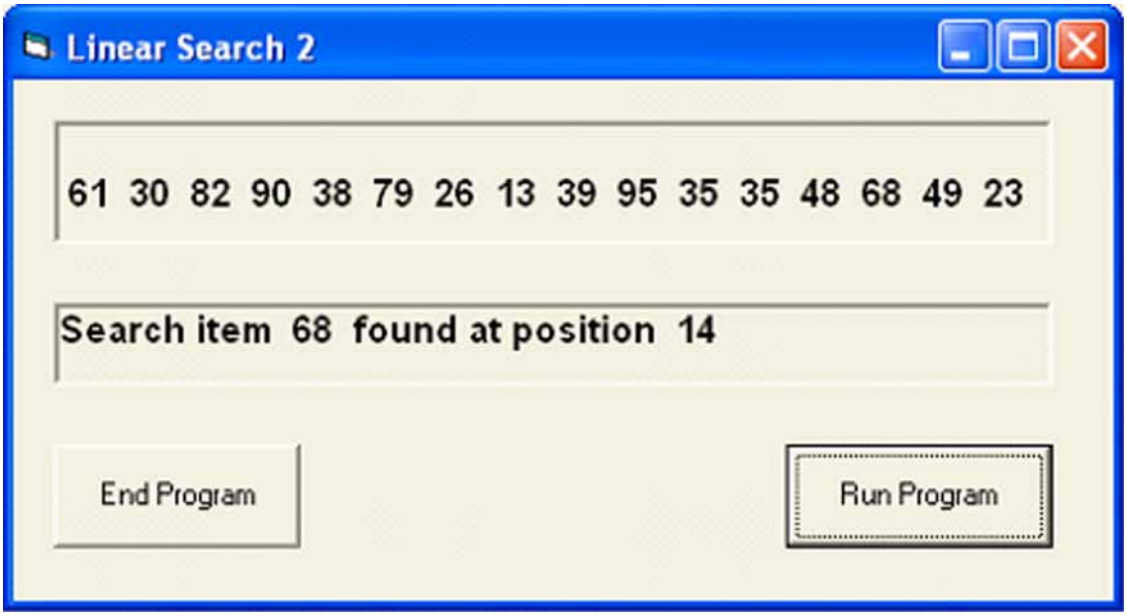


Figure 8.8:

The linear search is not a very efficient strategy since each array element has to be compared with search key until a match is found. For example it can be shown that for an array holding 1000 items an average of 500 comparisons will be made.

If the data items are ordered, however, then the search time can be improved but in this instance a binary search is significantly faster.

Despite the large number of comparisons required to find an entry it is probably the fastest technique for small arrays. It is also the most simplistic in terms of coding. It may also be the only method to search larger, unordered tables of data.

Searching a list of names


30 min

Write a program that will search a list of names for a given name. You should consider the following tests where the name:

- does not appear in the list
- appears at the start of the list;
- appears at the end of the list;
- appears within the list.

8.6.2 Counting Occurrences

Programs often have to count occurrences. Examples include counting the number of:

- students who achieved particular marks in an examination
- rainfall measurements greater than a particular level
- words equal to a given search value in a text file.

The basic mechanism is simple:

1. a counter is established
2. a list is searched for the occurrence of a search value
3. every time the search value occurs, the counter is incremented

You should notice that counting occurrences examines the entire list and so the algorithm is a variation of the linear search algorithm described above. The general counting occurrences algorithm is:

Counting occurrences - general algorithm

```
1.  counter = 0
2.  prompt the user for the search value
3.  set pointer to start of list
4.  do
5.    compare search item to list(position)
6.    if equal then
7.      increment count
8.    end if
9.    move to next position in the list
10. until until counter > 15
11. report number of occurrences
```

Below is the full Visual Basic implementation.

```
Option Explicit
Dim SearchKey As Integer, Pointer As Integer, Fill As Integer
Dim Occurrence As Integer
Dim List(15) As Variant
Private Sub Command1_Click()

'Program Counting Occurrences
'29th February 2004
'Program by Fred Fink

'This program will perform a search on
'an array holding 16 random integers and
'output occurrences of an input value

Setup                                                 'Call procedures
Populate_List List()
Enter_search_item SearchKey
Search_for_item Pointer, SearchKey, Occurrence, List()

End Sub


Private Sub Setup()                                 'Initialise variables
  Randomize                                         'and clear output boxes
  PicList.Cls
  PicResult.Cls
  PicList.Print
  Pointer = 0
  Occurrence = 0
End Sub

Private Sub Populate_List(ByRef List())           'Fill array with
  Dim Fill As Integer                             '16 random numbers
  For Fill = 0 To 15
    List(Fill) = Int(49 * Rnd) + 1
    PicList.Print List(Fill);
  Next Fill
End Sub

Private Sub Enter_search_item(ByRef SearchKey) 'User input search item
  SearchKey = InputBox("Input search item to count")
End Sub

Private Sub Search_for_item(ByVal Pointer, ByVal SearchKey,
ByVal Occurrence, ByRef List())
  Do
     If List(Pointer) = SearchKey Then
     Occurrence = Occurrence + 1
```

```
        End If
        Pointer = Pointer + 1
    Loop Until (Pointer > 15)
    PicResult.Print "Search item "; SearchKey; " found "; Occurrence;
"times in the list."
End Sub

Private Sub Command2_Click()
    End
End Sub
```

This file (Occurrences.txt), can be downloaded from the course web site.

In this case the array is filled with values from 1 to 49 in order to increase the chances of multiple occurrences.

The output of this can be seen in Figure 8.9

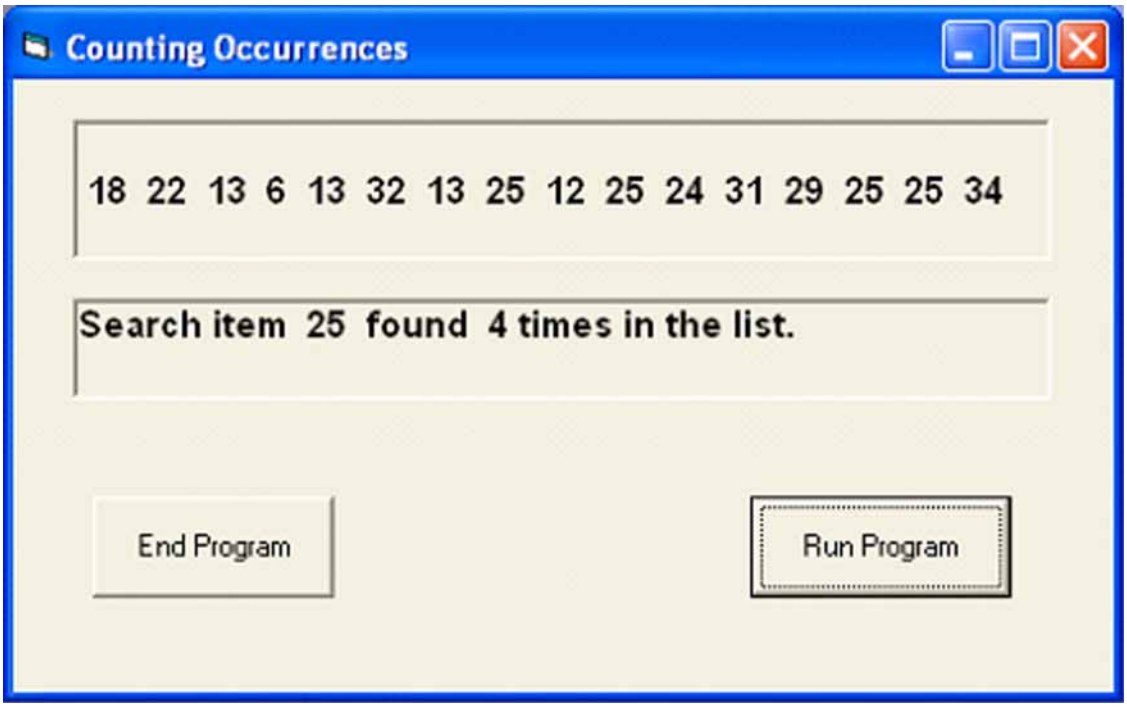


Figure 8.9:

Counting names in a list

Write a program that will count the number of times a search name appears in a list of names. Use the algorithm you have seen in the counting occurrences animation to help you. Remember to construct appropriate test cases in advance of coding. You should consider tests which:

- check for boundary conditions e.g. occurrence of the search value at the beginning or end of the list;
- no occurrences of the search value within the list;

- a single occurrence of the search value;
- multiple occurrences of the search value;

8.6.3 Finding Maximum and Minimum

Computers are often used to find maximum and minimum values in a list. For example, a spreadsheet containing running times for videos might make use of a maximum algorithm to identify the video with the longest running time, or a minimum algorithm to identify the shortest running time. A database containing personal details of club membership might make use of maximum and minimum algorithms to identify the oldest or youngest member. We are certain you can think of a few more for yourself. Clearly these algorithms are extremely useful and very widely used.

To find a maximum, we set up a variable which will hold the value of the largest item that has been found so far, usually the first data element. If an element in the array exceeds this working maximum, we give the working maximum that value.

Such algorithms sometimes have to return the index number of the largest or smallest element, and sometimes the actual maximum or minimum value. The algorithms to return the maximum and minimum values are shown below:

Finding the maximum

```
1. set maximum to first item in the list
2. set current position to 2
3. do
4.   compare maximum to item at current position
5.   if maximum < item then
6.      set maximum to item
7.   set current position to next position in the list
8. loop until end of list
9. report maximum value
```

Finding the minimum

```
1. set minimum to first item in the list
2. set current position to 2
3. do
4.   compare minimum to item at current position
5.   if minimum > item then
6.      set minimum to item
7.   set current position to next position in the list
8. loop until end of list
9. report minimum value
```

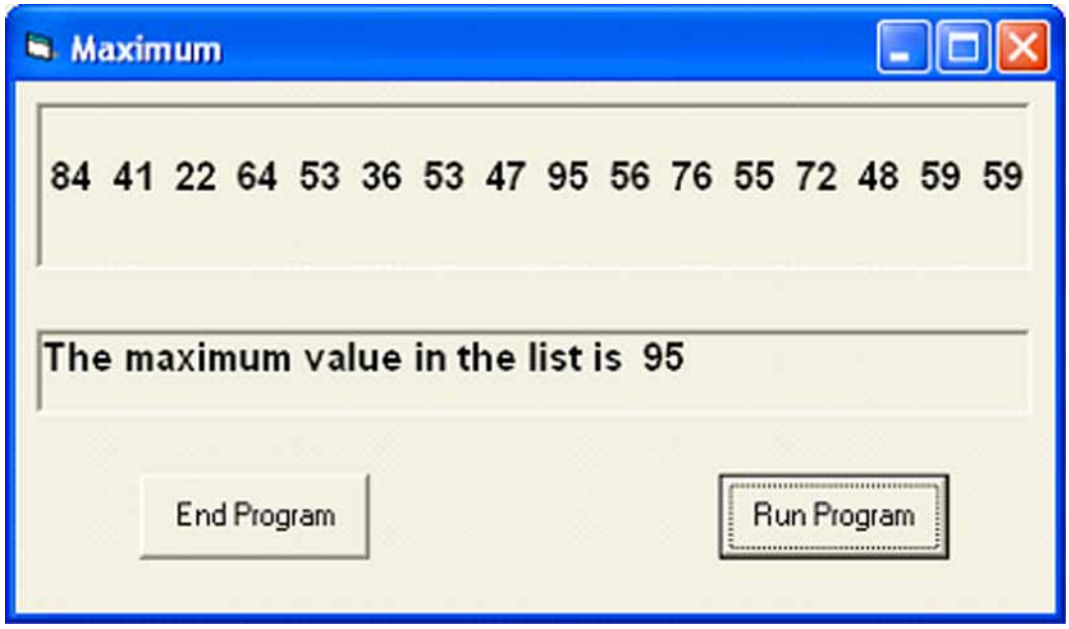
The full Visual Basic program for maximum is shown below:

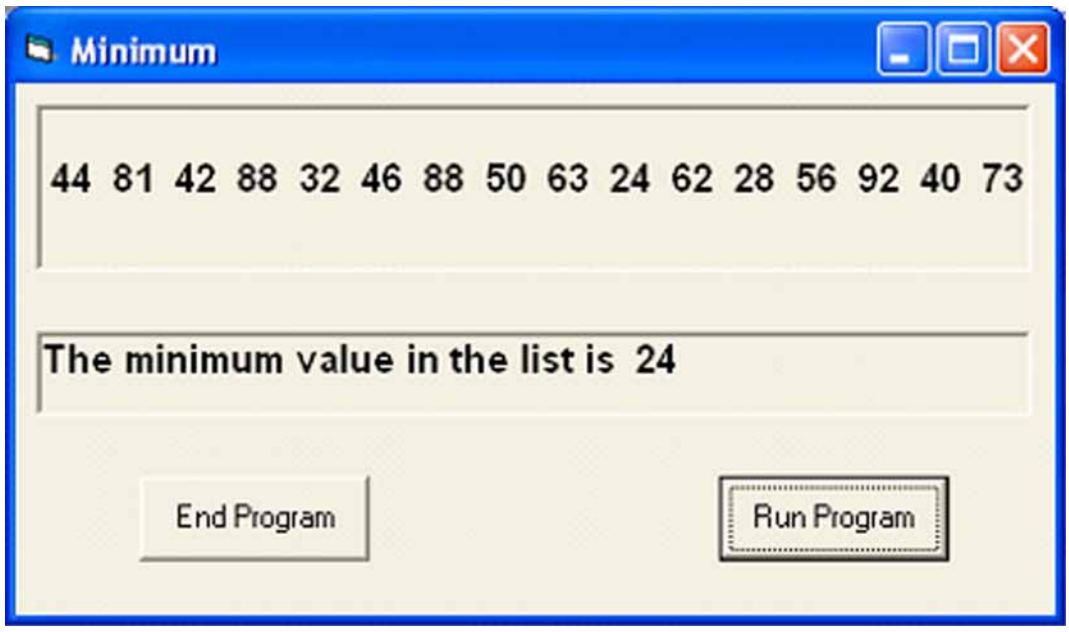
```
Option Explicit
Dim SearchKey As Integer, Pointer As Integer, Fill As Integer
Dim Maximum As Integer
Dim List(15) As Variant
Private Sub Command1_Click()

'Program Maximum
'29th February 2004
'Program by Fred Fink

'This program will perform a search on
'an array holding 16 random integers and
'output the maximum value

Setup                                              'Call procedures
Populate_List List()
Search_for_max Pointer, Maximum, List()

End Sub


Private Sub Setup()                              'Initialise variables
  Randomize                                          'and clear output boxes
  PicList.Cls
  PicResult.Cls
  PicList.Print
  Pointer = 0
  Maximum = 0
End Sub

Private Sub Populate_List(ByRef List())         'Fill array with
  Dim Fill As Integer                                      '16 random numbers
  For Fill = 0 To 15
    List(Fill) = Int(100 * Rnd) + 1
    PicList.Print List(Fill);
  Next Fill
End Sub


Private Sub Search_for_max(ByVal Pointer, ByVal Maximum, ByRef List())
  Maximum = List(Pointer)
  Do
     Pointer = Pointer + 1
     If Maximum < List(Pointer) Then          'find maximum
       Maximum = List(Pointer)
     End If
  Loop Until (Pointer > 14)
  PicResult.Print "The maximum value in the list is "; Maximum
```

```
End Sub

Private Sub Command2_Click()
  End
End Sub
```

This file (Maximum.txt), can be downloaded from the course web site.

The above code can easily be modified to return the minimum value.





Finding the maximum and minimum value in a list of integers

Write a program that will find the maximum and minimum values in a list of integers. Use the algorithm you have seen in the maximum/minimum animation to help you. Remember to construct appropriate test cases in advance of coding. You should consider the following tests:

- a normal test where maximum and minimum are as expected;
- a test where the maximum and minimum values are the same.

Sentence completion - algorithms

On the Web is a sentence completion task on algorithms. You should now complete this task.

8.7 Summary

The following summary points are related to the learning objectives in the topic introduction:

- understand and exemplify modularity in programming;
- understand and exemplify user-defined procedures and functions in programming;
- understand procedure/function parameters and how they are used;
- understand and exemplify parameter passing methods;
- understand and exemplify the following standard algorithms:
    1. linear search;
    2. counting occurrences;
    3. finding maximum/minimum.

8.8 End of topic test

An online assessment is provided to help you review this topic.

Glossary

Acceptance testing

Testing of software outside the development organisation and usually at the client site.

Adaptive maintenance

Takes place when a program's environment changes, for example a different operating system.

Algorithm

A detailed sequence of steps which, when followed, will accomplish a task.

Alpha testing

Testing of software within the development organisation.

Beta testing

Testing of software outside the development organisation using clients or selected members of the public.

Bottom-up design

A method of program refinement that starts with individual modules and builds them up into a complete program.

Boundary testing

Running a program with test data that represents the extreme upper and lower values. Within this range the program should operate normally.

Bugs

A bug is a program error.

Bytecode

This is produced by JavaScript and is a form of machine code that runs under the Java virtual environment. The latter is freeware and enables any computer to run Java programs

Client

The person or group that initiates the development process by specifying a problem .

Compiler

A program that translates a complete high level language program into an independent machine code program.

Concatenation

Joining of Visual Basic string variables to make longer strings using the '&' operator.

Corrective maintenance

Correction of previously undetected errors during development that is now apparent after installation of the software on the client site.

COTS

Commercial Off The Shelf software. An alternative software development system that allows programmers to purchase ready-made software. Can be an expensive option.

Data

Unstructured information. A collection of numeric or alphanumeric characters which can be processed by a computer. Raw data is meaningless to people.

Database

An organised and structured collection of related data.

Data modelling

A process used in object oriented languages that identifies objects, how they relate to one another and their manipulation.

Debugging

The detection, location and removal of errors in a program.

Declarative language

Programmers use this type of language to specify what the problem is rather than how to solve it by writing code. The language uses facts and rules to express relationships.

Desk checking

Akin to a dry run where the running of a program is checked without a computer.

Development team

Generic description of the personnel involved in developing the software solution.

Dry run

A pen and paper exercise to debug a program.

Event driven

A system that responds to an external event such as mouse click or a key press.

Event driven language

An event driven language that is designed to handle external events like interrupts, mouse clicks etc

Exceptions testing

Testing the robustness of a program by entering silly data - character data instead of numeric data, excessive values etc.

Executable code

Independent machine code that can be run without translation.

Exhaustive testing

Complete testing of a program under every conceivable condition. An expensive method time-wise.

Explicit declaration

Each variable, for example is declared unambiguously by the user so there is much less room for error in running programs Visual basic.

Feedback

A looping system where information is fed back in to a computer system. Previous output becomes new input.

Fit for purpose

The finished program runs to specification and is robust and reliable.

Function

A block of code like a procedure but a value is returned when the function is used.

Functional language

A language that utilises the evaluation of expressions rather than the execution of commands. It is based on the use of functions from which new functions can be created.

Functional specification

This will detail how the developed program will behave under specified conditions.

General purpose language

The language can be used to program solutions covering a broad range of situations.

High-level language

A language designed to be easily understood by programmers. They use commands and instructions based on English words or phrases.

Human computer interface

Allows the program to interact with the outside world. The interface is the only part of the program that users see.

Implicit declaration

If a variable, for example is not fully declared by the user then it is given default attributes by the Visual Basic language.

Independent test group

Testing of software by a group out with the development team.

Inheritance

The sharing of characteristics between a class of object and a newly created sub class. This allows code re-use by extending an existing class.

Intermediate code

A form of compiled code that is specifically produced for a target computer.

Internal commentary

The use of comments within source code to describe what it does.

Internal documentation

The use of comments within source code to describe what it does.

Interpreter

A program that translates a high level program line by line, which it then tries to execute. No independent object code is produced.

Iterative

An iterative process is one that incorporates feedback and involves an element of repetition.

Jackson Structured Programming

A diagrammatic design method for small programs that focuses on sequence, selection and iteration.

Java

A language designed by Sun Microsystems. The language is portable because Java interpreters are available for a wide range of platforms.

Keyword

A reserved word with a special meaning in a computer language. For example for, if, dim in Visual Basic.

Legal contract

A contract set up between client and development team, the details of which are set out in the requirements specification which becomes legally binding should anything go wrong.

Lexical analysis

Part of the compilation process where the source code is tokenised into symbols and stored in the symbol table.

Linear search

A standard algorithm that perform a sequential search on a list of data items.

Machine code

Native computer code that can be understood without translation.

Macro

A block of code that automates a repetitive task. Rather like a batch file they are normally created within an application then run by activating a key press combination or clicking on an icon.

Maintenance

The upkeep of a program by repair and modification.

Methodology

A technique involving various notations that enables the design of software to be implemented.

Module library

A module library includes code for standard algorithms that can be re-used by programmers.

Normal operation

Running of a program under expected normal conditions.

Object

A data item that can be manipulated by a computer system, for example a database record or a file.

Object oriented design

A method that centres on objects and the operations that can be performed on them.

Object-oriented language

An object-oriented computer language like Java that uses objects rather than actions and data rather than logic. An object is represented by a class that can be extended to involve inheritance.

Optimised

Refinement of code to make it more efficient.

Parameter

An argument of a procedure or function that represents a local variable.

Parameter passing

The mechanism by which data is passed to and from procedures and the main program.

Perfective maintenance

Takes place when a system has to be enhanced in some way e.g. program run faster.

Portable

The ability of a program to run on different machine architectures with different operating systems.

Problem oriented

The focus is on the problem and how it is to be solved rather than on the hardware on which the program will run.

Problem specification

A document outline of what is to be solved in terms of programming a solution to a given problem.

Procedural language

Also known as imperative languages because the programs follow a sequence of steps until they terminate. The code is made up of procedures and functions.

Procedure

A block of code that, when called from within a program will perform a specific action.

Process

An activity that is performed by a piece of software,

Programming team

A section of the development team responsible for the coding, testing, implementation and maintenance of the software.

Project manager

A member of the development team who is responsible for the supervision of the project. The main tasks are to keep the project on schedule and within budget.

Pseudo-code

A notation combining natural language and code used to represent the detailed logic of a program i.e. algorithmic notation.

RAD

Rapid Application Development. An alternative software development model that uses event driven languages for its implementation.

Recursion

A programming technique that is iterative in that a procedure or function can call itself. It is very demanding of computer memory.

Reference parameter

Here the address of the actual parameter is accessed by the formal parameter. Information is passed OUT from the procedure to the main program.

Reliable

A program is reliable if it runs well and is never brought to a halt by a design flaw.

Repetition

A process that repeats itself a finite number of times or until a certain condition is met.

Requirements specification

A document describing what the system must be able to do in order to meet user requirements.

Robust

A program is robust if it can cope with problems that come from outside and are not of its own making.

Scripting language

Used for writing small programs or scripts that enhances existing software. The best example is JavaScript which is used to enhance web pages.

Semantics

Semantics is the meaning of a statement in a given language.

Simulation

Replication of a process by computer that would not be possible to do manually. For example studying the projected traffic analysis of an airport or throwing a die many hundreds of times.

Software development environment

The high level language programming environment that offers tools and techniques to design and implement a software solution.

Software development process

A series of stages involving defined methods to produce a software project according to an initial specification.

Software engineering

A sphere of computing where the emphasis is on the development of high quality, cost effective software produced on schedule and within agreed costs.

Source code

The code for a program written in a high level language. This code is then translated into machine code.

Special purpose language

Languages designed for specific tasks such as prolog for artificial intelligence or C for writing operating systems.

Specification

A document outlining the program requirements set by the client.

SSADM

Structured Analysis and Design Model. An alternative to the waterfall model that deals only with the analysis and design phases of software development.

Standard algorithm

An algorithm that appears over and over again in many programs. Also called common algorithms.

Stepwise refinement

Similar to top-down design of sectioning a large and complex system into smaller and more easily manageable components.

Structure charts

A diagrammatic method of designing a solution to solve a software problem.

Structured data

Data that is organised in some way, for example an array or database.

Structured listing

Program listing clearly showing the modules involved complete with commentary and meaningful variable and procedure names.

Stub

A temporary addition to a program used to assist with the testing process.

Symbol table

Part of the compilation process where the tokens created by the lexical analysis phase are stored.

Syntax

Syntax means structure or grammar of a statement in a given language

Systems analyst

The person responsible for analysing and determining whether a task is suitable for pursuit using a computer. They are also responsible for the design of the computer systems.

Systems developer

Another name for a systems analyst.

Systems specification

An indication of the hardware and software required to run the developed program effectively. It will be the basis of subsequent stages which prepare a working program.

Technical guide

Documentation intended for people using a system containing information on how to install software and details system requirements such as processor, memory and backing storage.

Test data

Data that is used to test whether software works properly and that it is reliable and robust.

Testing

Running a program with test data to ensure a program is reliable and robust.

Test log

A record of how a program responds to various inputs.

Test plan

A strategy that involves testing software under verifying conditions and inputs.

Top-down design

A design approach of sectioning a large and complex system into smaller and more easily manageable components.

Trace facility

A method used to debug a program by tracing the change in values of the variables as the program is run.

Traditional model

An alternative name for the waterfall model that details the seven stages of program development.

Unusual user activity

Running a program with exceptional data.

User guide

A document intended for people using a system containing information on how to use the software.

Value parameter

Here a copy of the actual parameter is passed in to the formal parameter. Information is passed IN to the procedure form the main program.

Waterfall model

One of the earliest models for software development that incorporates 7 stages from analysis to implementation and maintenance.

Hints for activities

Topic 6: High Level Language Constructs 1

Calculating minutes

Hint 1: This problem naturally breaks up into four sections:

1. declare the variables;
2. get input;
3. calculate the minutes;
4. output results.

The input must consist of three numbers, the days, the hours and the minutes. So you need three variables to hold these figures. What will you name them? What type will they be (integer, real...)?

How do you work out the total number of minutes from these figures? You have to design a section of code to do this

Hint 2: There are 60 minutes in one hour and 24 hours in a day, hence 720 (24*60=1440) minutes in a day.

Hint 3: For example: days = 1, hours = 12, minutes = 15,

Total number of minutes = 2175.

days = 3, hours = 4, mins = 5

total minutes = 4565

days = 17, hours = 10, mins = 0

total mins = 25080

Calculating the number of digits in a number

Hint 1: It helps the user to know what type of number to type in, and what range of numbers is allowable *before* they type the wrong input! This should be in addition to any checks you do on the length of the number.

Calculating Leap Years

Hint 1: shows several years and indicates whether they are leap years or not. This data could be used to test your program. Table 8.2

Table 8.2: Calculation of leap years

Year	Is a Leap Year?
1900	No
1996	Yes
2000	Yes
2001	No

Answers to questions and activities

2 Features of Software Development Process

Revision (page 11)

Q1: c) Design, implementation, documentation, evaluation

Q2: b) Certain stages of the process are re-visited to make sure all is well

Q3: d) All of these

Q4: a) Plenty of colour

Q5: b) It is part of the design stage of the software development process

Answers from page 14.

Q6: c) The evolution of software systems were disorganised

Q7: d) Analysis, design, testing, evaluation

Q8: a) The model will be more realistic

Q9: d) All of the above

Q10: b) To enable personnel to discuss progress

Answers from page 16.

Q11: a) Top-down design

Q12: c) requirements specification

Q13: d) All of the above

Q14: b) Robust

Q15: a) A description of how the program should operate

Answers from page 20.

Q16: c) Testing finds faults and debugging removes them

Q17: c) It makes the program run more reliably

Q18: b) The final program meets the original specification

Q19: d) All of these

Q20: a) Structured program listing, installation guide, user guide, technical guide

Answers from page 25.

Q21: c) It can make running a program a less irritable experience

Q22: b) The requirements

Q23: c) Assembler

Q24: d) All of these

Q25: a) To allow for faster processing

Answers from page 29.

Q26: b) Testing is done within the organisation

Q27: c) No computers are involved

Q28: a) The program is tested by the clients

Q29: b) You cannot account for all possible errors

Q30: a) The input of silly data

3 Design notation, data flow and evaluation

Revision (page 43)

Q1: c) It is very useful in complex program designs

Q2: d) They can reduce the overall development time of a program

Q3: b) English and high level language code

Q4: c) Maintenance

Q5: a) An input error

Answers from page 52.

Q6: d) All of the above

Q7: b) It is breaking complex problems down into smaller units

Q8: d) All of the above

Q9: a) Thinks more about the solution to the problem

Q10: c) Makes the process more complex to novices

Answers from page 56.

Q11: a) To determine that the system meets the specification

Q12: b) Output statements at key points in the code

Q13: b) Testing can never show that a program is correct

Q14: c) Design input routines that will not crash when presented with unexpected data

Q15: a) They could remain hidden until the program is run under all conditions

Answers from page 60.

Q16: b) An error generated by the source translator

Q17: a) Do all program loops terminate?

Q18: d) All of the above

Q19: c) Logic error

Q20: b) De-bugging

Answers from page 63.

Q21: c) The program can cope with mistakes that the user might make

Q22: d) All of the above

Q23: b) A printer jam

Q24: b) Produce output, come what may

Q25: c) Java

4 Personnel

Answers from page 73.

Q1: c) The group who will purchase the software

Q2: d) Benefit the organisation in some way

Q3: d) The systems analyst is responsible for the entire project

Q4: a) Allow the systems analyst to produce a clear specification of the problem

Q5: b) The project manager

Answers from page 76.

Q6: d) All of the above

Q7: a) They report directly to the project manager at all stages of programming

Q8: b) Programmers will tend to test only within the functionality of their own code

Q9: c) The systems analyst

Q10: c) Testing is done by external groups on a variety of computer platforms

5 Languages and Environments

Revision (page 81)

Q1: d) All of the above

Q2: a) A compiler produces object code whereas an interpreter is much faster

Q3: d) Syntax error

Q4: c) Prolog is a procedural language used in artificial intelligence

Q5: c) A scripting language has to be translated

Answers from page 86.

Q6: b) Languages have to be adapted so new versions are released

Q7: c) Basic, Algol, Pascal, Comal

Q8: d) Moderation

Q9: b) C

Q10: d) All of these

Answers from page 95.

Q11: a) Programs have no pre-defined pathway

Q12: d) All of the above

Q13: b) Ease of developing of interactive web pages

Q14: c) Macros automate repetitive tasks

Q15: d) Changing audio CDs

Answers from page 99.

Q16: c) Java

Q17: d) All of these

Q18: c) It can promote code re-use by extending an existing class

Q19: a) Lisp

Q20: c) It emphasises the evaluation of expressions

Answers from page 103.

Q21: b) Lexical analysis

Q22: d) All of these

Q23: c) A compiler can produce more efficient object code

Q24: a) Looping structures have to be interpreted each time they are entered

Q25: a) Computers can only understand machine code

6 High Level Language Constructs 1

Revision (page 107)

Q1: b) The FOR..NEXT loop

Q2: d) Answer = 5 + 8 * (3 - 2)

Q3: b) NOT 1 = FALSE

Q4: c) 6

Q5: d) All of the above

Answers from page 123.

Q6: c) 17.5%Vat

Q7: a) It is easier for programmers to locate and fix errors

Q8: d) Integer

Q9: a) 1 or 2

Q10: c) Integer (long)

Practice in using simple variables (page 133)

Q11: The variable `number2` has not been declared. There will be no output since the variable sum has no value and has not been declared. Also the `Print` statement is wrong

Answers from page 137.

Q12: c) Global

Q13: d) All three statements above

Q14: b) They are hidden from other procedures and functions

Q15: c) The extent to which the variable can be 'seen' by the rest of the program

Q16: d) It can have only the values true or false

Answers from page 156.

Q17: d) NOT

Q18: a) 11

Q19: c) The execution of program statements in order, from beginning to end

Q20: b) The expression is TRUE when both conditions being tested are TRUE

Q21: c) Assignment

7 High Level Language Constructs 2

Revision (page 173)

Q1: c) Data items of the same type are grouped together

Q2: c) Only the first array element is set to some value

Q3: d) Days(3) = "Wednesday"

Q4: c) 1, 8, 3, 9, 5, 6

Q5: c) Iteration

Answers from page 184.

Q6: d) Control loop variable is increasing in value by a constant amount determined by the programmer

Q7: d) For loopCounter = (3*4.16) to (5*5.6) step 1

Q8: a) Calculating the total number of marks entered at a keyboard

Q9: a) Triangle

Q10: c) 18

Answers from page 192.

Q11: The value of `n` within the loop is never incremented. It will therefore always have a value of 0, which is less than 100, and hence this will produce an infinite loop which will never terminate. The program will therefore never end.

The code could be altered to include an increment, i.e.

```
n = 0
do while n < 100
  value = n*n
  n = n + 1
loop
```

Q12: There is no `loop` to end the `do..while`

Add loop at the end of the code:

```
i = 1
do while i <= 10
  print(i)
  i = i + 1
loop
```

Q13: A typical solution to the problem is shown here.

```
dim i as integer, sum as integer
sum = 0
i = 0
do while i <= 20
  sum = sum + i
  i = i + 1
loop
```

You should obtain a value of 210 for the sum of all numbers between 0 and 20 inclusive.

Answers from page 198.

Q14: c) The loop need not be entered if the condition fails at the start

Q15: a) Selection

Q16: b) 1 2 3 4 5

Q17: b) 1

Q18: d) Any one of the above

Indexing arrays (page 206)

Q19: c) array = [0, 3, 0, 0]

Q20: d) array = [7, 3, 0, 0]

Q21: c) 2

Q22: a) 3

Q23: c) array = [8, 12, 5, 19, 3, 0, 7, 52]

Q24: c) array = [8, 23, 5, 9, 3, 4, 7, 52]

Q25: array = [0, 9, 3, 0]

Q26: array = [0, 7, 0, 0, 6, 0, 0, 3, 2, 0]

Q27: The program fragment which will help you to do this is:

```
for i=0 to 9
  myarray(i) = i
```

Answers from page 208.

Q28: d) All of the above options

Q29: b) A for loop

Q30: d) 0 4 6 2 5 7

Q31: a) value_1(0) = "Hello"

Q32: b) 10

8 Procedures and Standard Algorithms

Revision (page 223)

Q1: b) That the input data is within specified limits

Q2: d) All of the above

Q3: b) 5

Q4: b) A sequence of instructions that can be used to solve a particular problem

Q5: a) Linear search

Answers from page 236.

Q6: b) Repetition of lines of code are avoided

Q7: d) A footer

Q8: d) All three options are true

Q9: b) Anything that happens to the formal parameter happens to the actual parameter

Q10: c) 5 6 15 3

Turning a procedure into a function (page 241)

```
function getValidNumber as integer
Dim numberOk as boolean
Dim number as integer
     do
        number = Inputbox("Input a number")
        numberOK = number > 0
        if not numberOK then
        Print("The number must be greater than 0")
        end if
     loop until numberOK
getValidNumber = number
end function
```

Using procedures or functions (page 242)

Using the procedures:

```
'program sum1
dim first as integer, second as integer

sub getValidNumber (ByRef number as integer)
var numberOk as boolean
```

```
      do
         number = Inputbox("Input a number")
         numberOK = number > 0
         if not numberOK then
         Print("Make it positive")
         end if
         loop until numberOK
end sub

function sum (a as integer, b as integer) as integer
     sum = a + b

     getValidNumber first
     getValidNumber second
     Print ("The total is "; sum(first, second))
end function
```

Using the functions:

```
'program sum2
dim first as integer, second as integer

function getValidNumber as integer
dim numberOk as boolean
dim number as integer
    do
        number = Inputbox("Input a number")
        numberOK = number > 0
        if not numberOK then
        Print("Make it positive")
        end if
    loop until numberOK
    getValidNumber = number
end function

function sum (a as integer, b as integer)as integer
     sum = a + b
end function
'main program
first = getValidNumber
second = getValidNumber
Print ("The total is "; sum(first, second))
```

Answers from page 247.

Q11: a) A function does not need to be declared

Q12: c) Square

Q13: b) y = ABS(-5)

Q14: d) A procedure or function can call itself

Q15: d) 12